Technical Report
CMU/SEI-92-TR-12
ESC-TR-92-012

(2)

Carnegie-Mellon University
Software Engineering Institute

Issues in Requirements Elicitation

Michael G. Christel
Kyo C. Kang

September 1992

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment or administration of its programs on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state or local laws, or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state or local laws, or executive orders. While the federal government does continue to exclude gays, lesbians and bisexuals from receiving ROTC scholarships or serving in the military, ROTC classes on this campus are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pa. 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pa. 15213, telephone (412) 268-2056.

Technical Report
CMU/SEI-92-TR-12
ESC-TR-92-012
September 1992

Issues in Requirements Elicitation

Michael G. Christel
Kyo C. Kang

Requirements Engineering Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESC/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.
This report was funded by the U.S. Department of Defense.

Table of Contents

List of Figures

List of Tables

Issues in Requirements Elicitation

Abstract: There are many problems associated with requirements engineering, including problems in defining the system scope, problems in fostering understanding among the different communities affected by the development of a given system, and problems in dealing with the volatile nature of requirements. These problems may lead to poor requirements and the cancellation of system development, or else the development of a system that is later judged unsatisfactory or unacceptable, has high maintenance costs, or undergoes frequent changes. By improving requirements elicitation, the requirements engineering process can be improved, resulting in enhanced system requirements and potentially a much better system.

Requirements engineering can be decomposed into the activities of requirements elicitation, specification, and validation. Most of the requirements techniques and tools today focus on specification, i.e., the representation of the requirements. This report concentrates instead on elicitation concerns, those problems with requirements engineering that are not adequately addressed by specification techniques. An elicitation methodology is proposed to handle these concerns.

This new elicitation methodology strives to incorporate the advantages of existing elicitation techniques while comprehensively addressing the activities performed during requirements elicitation. These activities include fact-finding, requirements gathering, evaluation and rationalization, prioritization, and integration. Taken by themselves, existing elicitation techniques are lacking in one or more of these areas.

1 Introduction

Requirements engineering is a key problem area in the development of complex, software-intensive systems [Brooks 87, p. 17]:

> *The hardest single part of building a software system is deciding what to build. ...No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.*

Issues involved in this problem area include:

- achieving requirements completeness without unnecessarily constraining system design
- analysis and validation difficulty
- changing requirements over time

Many requirements errors are passed undetected to the later phases of the life cycle, and correcting these errors during or after implementation has been found to be extremely costly [Congress 90]. The DoD Software Technology Plan [DoD 91] states that "early defect fixes are

typically two orders of magnitude cheaper than late defect fixes, and the early requirements and design defects typically leave more serious operational consequences." One way to reduce requirements errors is by improving requirements elicitation, an activity often overlooked or only partially addressed by current requirements engineering techniques.

Before proceeding further, a few definitions taken from the literature will help to clarify concepts related to requirements engineering. After the role of elicitation in the requirements engineering process is defined, an outline of the remainder of the report will be presented.

1.1 Definitions

A requirement is a "function or characteristic of a system that is necessary...the quantifiable and verifiable behaviors that a system must possess and constraints that a system must work within to satisfy an organization's objectives and solve a set of problems" [STEP 91]. Similarly, "requirement" has the following definitions [IEEE 90]:

> *(1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; (3) a documented representation of a condition or capability as in (1) or (2).*

Requirements do not only consist of functions, a misconception introduced in part because the currently popular structured analysis techniques focus on articulating functional requirements. Different authors will present different definitions, but there are clearly nonfunctional requirements as well as functional requirements. In one source, requirements are classified as follows [Southwell 87]:

1. functional requirements
2. nonfunctional requirements
    a. performance/reliability
    b. interfaces
    c. design constraints

Ashworth classifies requirements into five broad categories [Ashworth 89]:

1. functions ("what")
2. data ("what")
3. nonfunctional requirements ("how well")
4. goals (defined to guide the system developer in achieving the implementation of the agreed user requirements)
5. implementation/design constraints (e.g., use COBOL)

Sage and Palmer classify requirements as technical system requirements, which are primarily functional requirements, and management system requirements, which include cost and time constraints as well as quality factors for requirements [Sage 90]. The IEEE Standard Glossary of Software Engineering Terminology defines five other types of requirements in addition to functional requirements: performance requirements, interface requirements, design requirements, implementation requirements, and physical requirements [IEEE 90].

The varying uses of "specification" contribute to the difficulty in understanding requirements engineering. For example, Zeroual defines the *requirements specification* process as two steps: eliciting the user's requirements, and representing the requirements thus obtained [Zeroual 89]. However, other sources use "specification" more traditionally, with requirements analysis consisting of three interrelated operations: requirements elicitation, *specification*, and validation [Kramer 88]. In this paper "specification" shall be used as in this second context to mean the representation of the requirements, and not the requirements definition activity. This is consistent with the definition of requirements specification as "a document that specifies the requirements for a system or component" [IEEE 90]. The requirements definition activity shall be referred to as "requirements engineering."

Requirements engineering is "the disciplined application of scientific principles and techniques for developing, communicating, and managing requirements" [STEP 91]. Similarly, Loucopoulos and Champion define requirements engineering as "the systematic process of developing requirements through an iterative process of analysing a problem, documenting the resulting observations, and checking the accuracy of the understanding gained" [Loucopoulos 89, p. 123]. Leite offers the following process definition for requirements engineering [Leite 87, p. 26]:

> *Requirements analysis is a process in which "what is to be done" is **elicited** and **modeled**. This process has to deal with difference viewpoints, and it uses a combination of **methods, tools, and actors**. The product of this process is a model, from which a document, called **requirements**, is produced.*

This component activity of requirements elicitation is the focus of this report.

Requirements elicitation was defined in a recent workshop as "the process of identifying needs and bridging the disparities among the involved communities for the purpose of defining and distilling requirements to meet the constraints of these communities" [SEI 91, p. 26]. Requirements elicitation serves as a front end to systems development. Requirements analysts, sponsors/funders, developers, and end users are involved with requirements elicitation to differing degrees, and thus requirements elicitation involves social, communicative issues as well as technical issues [Zucconi 89], [Zahniser 90].

1.2 Report Outline

Many of the problems in creating and refining a system can be traced back to elicitation issues. Yet, much of the research conducted on requirements engineering has ignored elicitation, leading Leite to claim in a 1987 survey on requirements analysis that the state of the art in re-

quirements analysis is not much different than it was in the late 1970s. He argues that there is a concentration of research in this area on precision, representations, modeling techniques, verification, and proofs as opposed to improving the elicitation of requirements. He concludes that "research efforts should be directed towards methods and tools needed to improve the requirements analysis process, and, in particular, to those providing more support to the elicitation of requirements" [Leite 87, p. 3].

The requirements elicitation process is discussed further in Section 2. Section 3 then outlines the problems with requirements engineering in general, followed by an examination of the deficiencies attributable to requirements elicitation. A number of elicitation techniques have been proposed or put into practice during the past decade, and these techniques are surveyed in Section 4. This survey reveals that none of these techniques comprehensively address all of the identified elicitation deficiencies. A methodology is proposed in Section 5 based on these techniques which better addresses the problems of elicitation.

2 The Process of Requirements Elicitation

Rzepka decomposes the requirements engineering process into three activities [Rzepka 89]:

1. elicit requirements from various individual sources;
2. insure that the needs of all users are consistent and feasible; and
3. validate that the requirements so derived are an accurate reflection of user needs.

This model implies a sequential ordering to the activities, with elicitation done once at the very beginning of the process. In reality, though, the process is iterative, with these activities revisited many times [Southwell 87, p. 195]:

> *...the requirements definition activity cannot be defined by a simple progression through, or relationship between, acquisition, expression, analysis, and specification. Requirements evolve at an uneven pace and tend to generate further requirements from the definition processes.*
>
> *The construction of the requirements specification is inevitably an iterative process which is not, in general, self-terminating. Thus, at each iteration it is necessary to consider whether the current version of the requirements specification adequately defines the purchaser's requirement, and, if not, how it must be changed or expanded further.*

Thus, while requirements elicitation consists of the earliest activities in the requirements engineering process, it can not be divorced from the subsequent activities. Elicitation will likely iterate with these other activities during requirements development.

Requirements elicitation itself can be broken down into the activities of fact-finding, information gathering, and integration. For example, Rzepka further decomposes the elicitation process as follows [Rzepka 89]:

1. Identify the relevant parties which are sources of requirements. The party might be an end user, an interfacing system, or environmental factors.
2. Gather the "wish list" for each relevant party. This wish list is likely to originally contain ambiguities, inconsistencies, infeasible requirements, and untestable requirements, as well as probably being incomplete.
3. Document and refine the "wish list" for each relevant party. The wish list includes all important activities and data, and during this stage it is repeatedly analyzed until it is self-consistent. The list is typically high level, specific to the relevant problem domain, and stated in user-specific terms.

4. Integrate the wish lists across the various relevant parties, henceforth called viewpoints, thereby resolving the conflicts between the viewpoints. Consistency checking is an important part of this process. The wish lists, or goals, are also checked for feasibility.
5. Determine the nonfunctional requirements, such as performance and reliability issues, and state these in the requirements document.

These activities are common to most of the process definitions for requirements elicitation found in the literature. However, the means of achieving these activities and iterating between them are still not well understood.

The resulting product from the elicitation phase is a subset of the goals from the various parties which describe a number of possible solutions. The remainder of the requirements engineering process concerns the validation of this subset to see if it is what the sponsor/funder and user actually intended. This validation typically includes the creation of models to foster understanding between the parties involved in requirements development. The result of a successful requirements engineering process is a requirements specification, where "the goodness or badness of a specification can be judged only relative to the user's goals and the resources available" [Fickas 88, p. 39].

The goal of requirements engineering is the production of a good requirements specification. The IEEE Guide to Software Requirements Specifications [IEEE 84] defines a good software requirements specification as being:

- unambiguous
- complete
- verifiable
- consistent
- modifiable
- traceable
- usable during operations and maintenance

Recent requirements engineering literature is in agreement on this set of attributes, with the added property that the requirements should be necessary [Cordes 89], [Schouwen 90]. The requirements should be prioritized as well, particularly in novel situations where the order in which the subgoals are addressed significantly impacts the final solution [Holbrook 90].

A good requirements elicitation process supports the development of a specification with these attributes. Conversely, problems with requirements elicitation inhibit the definition of requirements which are unambiguous, complete, verifiable, consistent, modifiable, traceable, usable, and necessary. Some of these problems are looked at in the next section. In subsequent sections the process sketched out here as "fact-finding, information gathering, and integration" will be refined to specifically address the problems encountered during requirements elicitation.

3 Requirements Elicitation Problems

Problems of requirements elicitation can be grouped into three categories:

- problems of scope, in which the requirements may address too little or too much information;
- problems of understanding, within groups as well as between groups such as users and developers; and
- problems of volatility, i.e., the changing nature of requirements.

The list of ten elicitation problems given in one source [McDermid 89] could be classified according to this framework as follows:

- problems of scope
  - the boundary of the system is ill-defined
  - unnecessary design information may be given
- problems of understanding
  - users have incomplete understanding of their needs
  - users have poor understanding of computer capabilities and limitations
  - analysts have poor knowledge of problem domain
  - user and analyst speak different languages
  - ease of omitting "obvious" information
  - conflicting views of different users
  - requirements are often vague and untestable, e.g., "user friendly" and "robust"
- problems of volatility
  - requirements evolve over time

The remainder of this section will discuss these three problem areas in further detail.

3.1 Problems of Scope

Elicitation techniques need to be broad enough to establish boundary conditions for the target system, yet still should focus on the creation of requirements as opposed to design activities. Avoiding contextual issues can lead to requirements which are incomplete, not verifiable, unnecessary, and unusable. Focusing on broader design activities improperly emphasizes developers' issues over the users' needs and may result in poor requirements as well.

Requirements elicitation must begin with an organizational and context analysis to determine the boundary of the target system as well as the objectives of the system. Less ambitious elicitation techniques not addressing this concern run the risk of producing requirements which are incomplete and potentially unusable, because they do not adhere to the user's or organi-

zation's true goals for the system. Performing an organizational and context analysis allows these goals to be captured, and then used to verify that the requirements are indeed usable and correct.

Elicitation techniques can be overly ambitious as well. Elicitation must focus on the creation of requirements, not design activities, in order to adequately address users' concerns and not just developers' needs. Elicitation strategies which produce requirements in the form of high level designs run the risk of creating requirements which are ambiguous to the user community. These requirements may not be verifiable by the users because they cannot adequately understand the design language. Also, requirements expressed as a design are much more likely to incorporate additional decisions not reflecting user or sponsor needs, i.e., the requirements will not be precise and necessary.

There are at least three broad contexts which affect the requirements and the requirements engineering process for a proposed system:

- organization
- environment
- project

Requirements elicitation necessitates an understanding of the organization in which the system under development will be placed, as well as an understanding of the system's mission within the organizational context: "the primary interest of customers is not in a computer system, but rather in some overall positive effects resulting from the introduction of a computer system in their environment" [Dubois 88, p. 395]. This view is not well reflected in current practice, where requirements elicitation concentrates on the computer system without much concern for organizational factors. Organizational factors include:

- submitters of input to the target system
- users of the target system's output
- ways in which the target system will change the organization's means of doing business

If requirements elicitation begins without an appreciation for organizational context, then a number of restricting assumptions are made due to "misconceptions, management politics, technical ignorance, mistrust, established practices, personnel resistance, ..." [Mittermeir 90, p. 121].

Environmental factors have a strong influence on requirements elicitation [Macaulay 90, p. 107]: "The process for eliciting the work-group and end-user requirements are premised on the notion that sound and accurate descriptions of the users and their environment is at first necessary." Environmental factors include:

- hardware and software constraints imposed on a target system (the target system will typically be a component of some larger system with an existing or required architecture already in place)

- the maturity of the target system's domain
- the certainty of the target system's interfaces to the larger system
- the target system's role within a larger system

Environmental constraints are introduced because the system under development is rarely a stand-alone system but instead must interface with a larger system. This premise allows the requirements engineer to restrict the requirements analysis to the universe of discourse established by this larger system [Leite 87].

Environmental constraints can have a profound impact on the requirements elicitation process. The specialization of this process to a particular architecture or domain allows requirements elicitation to focus on problems that either have lower priority or do not exist in other application domains [Leite 87, p. 60]: "performing requirements analysis for an application area may require specific methods and tools that are not necessary for other types of application."

Indeed, specialized methods for the embedded real-time systems domain have been proposed [Lavi 88], [Winokur 90]. Stair and LaMothe suggest that eliciting requirements for a real-time system will require different approaches than eliciting requirements for a batch transaction processing system [Stair 91].

The project context also affects the requirements and requirements engineering process. Project factors include:

- the attributes of the different stakeholder communities, such as the end users, sponsors, developers, and requirements analysts. Examples of such attributes are:
  - management style
  - management hierarchy
  - domain experience
  - computer experience
- the constraints imposed by the people involved in the elicitation process, e.g., managerial constraints concerning cost, time, and desired quality in the target system

The understanding of organizational, environmental, and project context thus provides a good starting point for requirements elicitation. Requirements engineering is supposedly bounded in the other direction by design activities, where a version of the requirements specification is first completed and then design for that specification takes place. In reality, though, high level design and requirements specification are done nearly simultaneously, i.e., you cannot separate "the How's from the What's" [Lavi 88]. Colbert notes that "lately, some have argued that there is *no* distinction between requirements analysis and design" [Colbert 89, p. 413]. He then presents an example of a problem requirement for "user-friendly interface" being expressed as a solution: "mouse-driven window environment." Colbert also discusses the distinction be-

tween requirements and design as being a matter of viewpoint. For example, a missile is a design to the person wanting a delivery system and a "requirement" to the person who created the missile.

While it is a noble goal to separate requirements elicitation from design activities, it may be difficult to achieve in practice. Achieving this separation allows requirements to be created which are both complete (full coverage of users' and other stakeholders' needs) and necessary (only represent information pertinent to solution development). However, typically the initial requirements are either underspecified, necessary and incomplete; or else they are overspecified, complete but burdened with needless design constraints. Cordes and Carver discuss this difficulty in producing requirements which are both necessary and complete [Cordes 89, p. 184]:

> *A true distinction between the processes involved in necessity testing and completeness testing requires knowledge about the actual implementation of the system. Only the knowledge that is present in the final, completed system product can determine if the inclusion of a piece of information is needed for specification completeness or if this information is unnecessary to the system's development.*

Thus, elicitation activities which are either too narrow or too broad in scope may result in requirements which are ambiguous, incomplete, not verifiable, unnecessary, and unusable. The requirements may be unusable because they do not reflect true user needs, or else because they are not implementable under given environmental or project constraints.

3.2 Problems of Understanding

A Savant Institute study found that "56% of errors in installed systems were due to poor communication between user and analyst in defining requirements and that these types of errors were the most expensive to correct using up to 82% of available staff time" [Goodrich 90, p. 202]. Problems of understanding during elicitation can lead to requirements which are ambiguous, incomplete, inconsistent, and even incorrect because they do not address the requirements elicitation stakeholders' true needs.

Problems of understanding can be separated into three issues:

- The communities involved in elicitation possess a variety of backgrounds and experience levels, so that which is common knowledge to one group may be completely foreign to another. This makes it difficult for a requirements analyst to interpret and integrate information gathered from these diverse communities.
- The language used to express the requirements back to these stakeholder communities may be too formal or too informal to meet the needs of each of the groups, again because of the diversity of the communities.

- The large amount of information gathered during elicitation necessitates that it be structured in some way. The understanding of this structure is dependent on the characteristics of the stakeholder communities.

The stakeholders involved in requirements elicitation come from at least five communities: customers/sponsors, users, developers, quality assurance teams, and requirements analysts. The requirements should be expressed in a form which:

- promotes communication and understanding between customers and users;
- allows the developer to determine whether the expressed requirements are implementable; and
- lets quality assurance teams verify that an implementation meets these requirements.

The stakeholder communities may be multilevel. The involved parties may be in managerial positions in a contributing organization, e.g., the manager of the user organization, or they may be the actual end users within that organization.

The party representing a stakeholder community, e.g., the customer, may be the same party representing another stakeholder, e.g., the user in a case where the user community is funding the project directly. Hence the customer, user, developer, tester, and requirements analyst groups may represent five different entities or be represented by fewer groups which cross disciplines.

Information gathered from only one group, or only one level, is likely to be biased by the level of abstraction from which those people conceive the problem, their planning horizon, detailed acquaintance with the application, personal preconceptions, goals, and responsibilities. Therefore, "the true picture of the 'problem to be solved' can be obtained only from collecting information from all parties concerned" [Mittermeir 90, p. 121]. This information gathering can be a difficult task. System developers and requirements analysts may have limited knowledge of the application domain, while the system users may not know design methods for the development of systems with a significant software component. The customer may not understand what can be done to solve a given problem, nor have full appreciation for the difficulty in getting the analyst and developer to understand the problem in the customer's domain. The analyst is often ignorant of the customer's problems, goals, and wishes. Sage and Palmer note that "any attention to the development of communication skills to alleviate these problem areas, is of great benefit" for requirements elicitation [Sage 90, p. 99].

Requirements elicitation starts with inherently informal knowledge and typically involves people not literate in software-intensive systems. It therefore is plagued with the problem of verification of information due to the misunderstanding between requirements analysts and customers [Dubois 88]. To avoid misunderstandings due to terminology differences, requirements have traditionally been expressed back to the elicitation communities using natural language text. However, this approach introduces other problems, such as ambiguity [Milsom 89, p. 137]: "The use of natural language to express requirements inhibits automated analysis and

verification of the requirements. Requirements specified using natural language can be misinterpreted." Requirements therefore may be difficult to understand by the elicitation communities because of the form used to express the requirements.

Requirements are typically not the product of a single person's work but rather are the result of many people's expressed needs across different communities. These multiple inputs present problems about redundancy of data, inconsistency, and point of view [Cordes 89, p. 182]. Different groups involved in requirements elicitation have different interpretations of what the requirements say and different expectations of what a system built to these requirements will deliver.

Requirements may also be misunderstood because of their large size [Kramer 88, p. 86]:

> *...requirements may be misunderstood because they are so complex that the client and practitioner have difficulty focusing attention on one aspect at a time and perceiving interactions between requirements, or because the specified system is impossible to visualize from the resulting specification.*

One reason why requirements elicitation methodologies have not been used frequently in the past on real world information systems is due to the problem of managing all of the collected data [Ceri 86]. A huge amount of documentation is typically required by many methodologies, leading to the observation that "in most cases the documentation is on paper modules, and thus becomes quickly unmanageable and not up-to-date" [Ceri 86, p. 207].

Problems in understanding result from the necessary involvement of requirements analysts, sponsors, developers, and end users in requirements elicitation. The requirements are produced and interpreted by people with different experience levels and backgrounds. The form in which the requirements are expressed and the size of the system described by the requirements also affect understanding. If the participants in elicitation do not adequately understand the output of the process, then the resulting requirements may be ambiguous, inconsistent, and incomplete. The requirements may also be incorrect, not addressing the true needs of the elicitation communities.

Requirements analysis is a social process, and "techniques and methods that fail to recognize this factor (most of them do fail!) have less chance of gathering all the necessary information" [Leite 87, p. 30]. In summary, "good communication among users, customers, and developers is very important in solving pragmatic system problems, an issue too often overlooked when system analysis is approached only from a computer-science standpoint" [Deutsch 88, p. 44].

3.3 Problems of Volatility

Requirements change. During the time it takes to develop a system the users' needs may mature because of increased knowledge brought on by the development activities, or they may shift to a new set of needs because of unforeseen organizational or environmental pressures.

If such changes are not accommodated, the original requirements set will become incomplete, inconsistent with the new situation, and potentially unusable because they capture information that has since become obsolete.

One primary cause of requirements volatility is that "user needs evolve over time" [Sage 90, p. 8]. The requirements engineering process of elicit, specify, and validate should not be executed only once during system development, but rather should be returned to so that the requirements can reflect the new knowledge gained during specification, validation, and subsequent activities. A requirements engineering methodology should be iterative in nature, "so that solutions can be reworked in the light of increased knowledge" [Macaulay 90, p. 102].

Another cause of requirements volatility is that the requirements are the product of the contributions of many individuals, and these individuals often have conflicting needs and goals. For example, there usually is more than one customer, with each customer having different and often contradictory views and interests [Dubois 88]. Due to political climate and other factors, the needs of a particular group may be overemphasized in the elicitation of requirements. Later prioritization of the elicitation communities' needs may correct this oversight and result in requirements changes. Both the traceability of requirements and their consistency may be affected if these changes are frequent and not anticipated.

Organizational complexity is another cause of requirements volatility. Organizational goals, policies, structures, and work roles of intended end users all may change during the course of a system's development, especially as the number of users affected by a system's development increases. An iterative process of requirements development can address the problems of volatility [Dobson 92]:

> *The traditional notion of the software development life-cycle with requirements capture being completed before the design stage is no longer satisfactory. Requirements capture and design are now seen to be symbiotic. The initial set of requirements needed to start off the design process is gradually refined into a systematic and coherent statement of requirements hand in hand with the refinement of design.*

Due to the problems of understanding and scope discussed earlier, user needs may not be clearly expressed initially in the requirements, and the developer or requirements analyst may make some incorrect assumptions based on this ambiguity. With an iterative process, those mistaken assumptions can be detected faster and corrected sooner. For example, an iterative process allows the user to receive feedback much sooner on the developer's interpretation of the requirements, and to then correct problems as they are found. Many traditional development approaches do not give one stakeholder community such as the user much feedback on other stakeholders' interpretations until the complete system is delivered. An iterative requirements engineering process is illustrated in Figure 3-1.

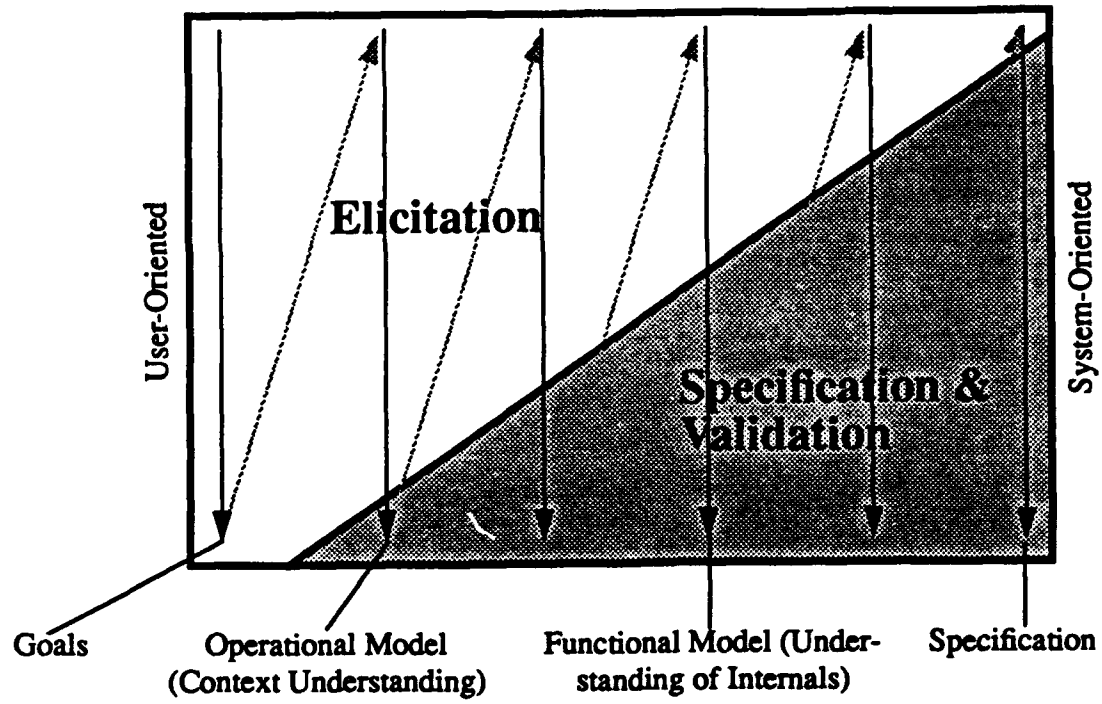

Figure 3-1 Requirements Engineering as an Iterative Process

A historical examination into the *IEEE Standard Glossary of Software Engineering Terminology* reveals an increasing awareness of the iterative nature of requirements development. In the 1983 glossary, "requirements analysis" is defined as "the process of studying user needs to arrive at a definition of system requirements" [IEEE 83]. This implies a one time, up front requirements definition activity. In the 1990 glossary, however, a second definition has been added for requirements analysis: "the process of studying and refining system, hardware, or software requirements" [IEEE 90]. This implies retrospective examinations of requirements with refinement steps, i.e., an iterative requirements engineering process.

Requirements are not completely known at the start of a system's development. They cannot be specified completely up front in one voluminous document, but rather will evolve during the analysis phases of a project and beyond. The communities involved in the elicitation, including users, developers, and customers, all learn and grow during the system's development and maintenance. This increasing knowledge possessed by the elicitation communities regarding the system should be utilized to improve the system, rather than prohibited because the requirements are to remain static.

4 Current Elicitation Techniques

Requirements elicitation has received little attention in the past from the software engineering research community [Leite 87, p. 2]:

> *From the survey, it was learned that requirements analysis, in particular requirements elicitation, is a hard task, and that it is carefully avoided by most of the software engineering researchers. We believe that most researchers avoid dealing with elicitation of requirements, because it is an area where one has to deal with informality, incompleteness and inconsistency. Instead, research labeled as dealing with requirements, usually deals with specification, and that is the main reason for the lack of agreement on the definitions of requirements analysis and specification.*

Despite this lack of research activity, there have been some efforts toward the development of methods and techniques to address the requirements elicitation problems discussed in the previous section. These approaches will be introduced here, along with a brief description of their proposed solutions to the problem areas of scope, understanding, and volatility. A few elicitation techniques receive more detailed treatment in the appendices.

The elicitation issues discussed in the previous section were categorized as follows:

- scope
  - organizational and contextual factors, including the identification of system goals, the problem context, and boundaries and interfaces
  - avoidance of premature design activities
- understanding
  - contribution of many varied stakeholder communities to elicitation
  - contribution of more than one person to requirements elicitation
  - large size of requirements and associated data resulting from elicitation
  - utility of multiple expressions (models) of the requirements
- requirements volatility

Table 4-1 illustrates the level of applicability of the techniques discussed in this section to these various issues, and thus indicates which elicitation problems a given technique or set of techniques address.

Table 4-1 also contains two columns on the properties of the techniques themselves, rather than the issues the techniques address. The column labeled "Technique Maturity" indicates the relative maturity of these techniques. Some of the techniques discussed in this section are very mature, having been tested and used in requirements elicitation for a decade or longer. Others are newly proposed, primarily research efforts, or have not yet been used much in the elicitation arena. A technique may receive no rating in this maturity column and still produce wonderful results; it just does not have a proven track record showing its applicability to elicitation. The column labeled "Defined, Prescriptive Technique" indicates the amount of guid-

Key ✔✔ - (1) technique strongly recognizes the issue & provides a means to deal with it; (2) possesses the given quality ✔ - (1) technique supports the issue, although perhaps indirectly or not as strongly as other techniques; (2) technique possesses the given quality to a limited extent No mark - technique does not address the issue at all or provides very little support for it	Organizational/Context Analysis	Avoidance of Premature Design	Various Contributors	Various Contributing Disciplines	Large Problem Size	Multiple Expressions	Volatility	Technique Maturity	Prescriptive, Defined Technique
Information Gathering									
Interviews	✔		✔	✔				✔✔	
Structured Interviews (e.g., via IBIS)	✔		✔✔	✔			✔	✔	✔✔
Team Approach (e.g., JAD)	✔✔		✔	✔✔				✔✔	✔
Use of Domain & Architectural Models	✔✔				✔	✔	✔		
Reqs. Expression & Analysis									
CORE	✔		✔	✔		✔		✔✔	✔✔
Delugach's Multiple Views	✔		✔✔	✔✔		✔✔			
SSM	✔						✔		✔
PDM	✔		✔	✔		✔	✔		✔
Validation									
QFD		✔			✔		✔		✔✔
Concept Prototyping					✔	✔	✔		
Iterative process use of elicit. techniques					✔		✔✔		

Table 4-1: Matching Elicitation Techniques to a Set of Issues

ance incorporated into the technique so that the requirements elicitor knows how to proceed. Some of the techniques have well defined steps and offer prescriptive advice. Others are generic and rely heavily on the skills and experience of the technique implementors for the technique to work effectively and efficiently in given elicitation situations.

4.1 Information Gathering

Interviews are perhaps the most common technique used for gathering information during requirements elicitation. There are many social aspects of dealing with users in interviews [Zucconi 89], [Berlin 89]. Berlin notes that "even a few hours of interviews can be very valuable,

even when conducted by engineers who have had only brief training in interviewing and user needs analysis" [Berlin89, p. 94]. The information collected through interviews can address organizational and contextual factors provided that the right questions are asked. Likewise, if the right people are interviewed the information will represent multiple stakeholders' opinions across a number of different communities affected by the development of the proposed system being elicited.

The organization and expression of the information collected through interviews is a neglected issue. There is a lack of standardized procedures for structuring information received via interviews: "in particular, during the interviews necessary to collect information, no procedure explains how the software analyst/tool documents the information, or determines the sequence of questions to ask" [Zeroual 89, p. 350]. Little tool support exists to help with the interviewing process as well: "it is difficult first to make it efficient and short, and secondly to automate the tasks it involves" [Zeroual 89, p. 350].

Other limitations with eliciting requirements primarily or exclusively through interviews result from the tremendous responsibility placed on the requirements analyst. Assuming that interview data was collected from the different communities affected by the system being elicited, the analyst must integrate these different interpretations, goals, objectives, communication styles, and use of terminology into a single set of requirements. This integration is a difficult task unless the interviews are structured in some way. For example, the use of a glossary of system-specific terms may reduce the number of inconsistencies in interviews that subsequently have to be resolved by the analyst.

Even with structured interview data, the analyst still must perform complex tasks such as deciding whether a particular piece of information is premature design information or really a requirement. These tasks require that the analyst is experienced in both the system domain and with development techniques, qualifications which may be difficult to satisfy.

With so much decision-making resting with the analyst, the elicitation stakeholders may not understand how the resulting requirements were derived and may refuse to share ownership in and approve these requirements. The requirements themselves may not be understandable, e.g., if written with a behavioral tone in very domain specific terms the users may comprehend everything but the developers could have difficulty. Finally, this integration and decision-making by the analyst takes time, and given that requirements are volatile, the longer this process takes the more likely it is that the subsequent requirements no longer match the stakeholder communities' needs and expectations.

Other techniques can be used in conjunction with interviews to help structure them and facilitate integration. The gIBIS method, highlighted in Appendix A.1, addresses social issues by inhibiting unconstructive rhetorical moves and supporting more constructive communication [Conklin 88]. However, gIBIS makes you think within a particular framework of issues, positions, and arguments, and this can be disruptive during the early phases of a design problem which is "critical and fragile and must be allowed to proceed in a vague, contradictory, and incomplete form as long as necessary" [Conklin 88, p. 325].

An advantage to having this structure of "issues, positions, and arguments" is that the analyst now has a means of integrating interview data, through the matching of issues. Also, through a comparison of this structure across different communities, the analyst can determine whether some individuals hid requirements in arguments for certain issues, believing that those requirements were obvious or implicit. Those requirements may have been obvious to that particular community, e.g., the end users, but perhaps need to be explicitly stated to another group such as the developers.

Domain models are another way to structure interviews or team approaches. The use of domain models, such as the feature oriented domain models discussed in Appendix A.2 and elsewhere [Kang 90], has many potential advantages for elicitation:

- improves understanding between multidisciplinary team by providing structure for vast amounts of information needed for the acquisition and expression of requirements; this results in better quality requirements and lower development costs
- builds a reusable base of domain knowledge to leverage from in future developments
- reduces ambiguity
- simplifies conflict detection by identifying issue points where conflict can arise
- promotes conflict resolution by providing rationale at issue points

Cameron notes that the animations of problem domain models have sometimes had a dramatic impact on the users' understanding of the models and on the whole process of elicitation of requirements [Cameron 89]. Despite this promise, however, not much use has been made of domain models for requirements elicitation. The use of feature oriented domain models in requirements elicitation has not yet been demonstrated or tested. Even if it is successful, the cost of developing domain models could be prohibitive.

Rather than just structuring interviews, another technique focuses on ensuring that information is gathered from all of the affected parties, and that the resulting requirements meet the approval and understanding of all of these parties, rather than just being the work of the requirements analyst. These team approaches to requirements elicitation make sure that issues of scope are properly addressed by getting the appropriate people involved at the very beginning of requirements elicitation. Likewise, they explicitly recognize that there are issues of understanding dependent upon the variety of disciplines and number of people affected by the development of a proposed system.

The team approach to requirements elicitation is characterized by JAD, an acronym for Joint Application Design. JAD focuses on improving the group process and getting the right people involved at the start [Zahniser 90]. This technique has been used successfully by IBM since the late 1970s, and its advantages include the promotion of cooperation, understanding, and teamwork among users, developers, and customers. Developers help users formulate problems and explore solutions, while users share ownership of the requirements and associated

documents. Through the use of structured meeting procedures, facilitation protocols, and visual aids, JAD enhances idea generation and evaluation, communication, and consensus generation. Guidance on using JAD is provided in [Wood 89], which emphasizes its use for online, transaction-based systems. JAD is also discussed further in Appendix A.3.

While this technique has been used successfully, a recognized problem is that all of the participants funnel their ideas through a facilitator or recorder. Thus, the recorder may inadvertently impose an interpretation on the collected data not shared by the group. For integration, JAD is dependent on the skills of the recorder, much as the integration of structured interviews is dependent on the skills of the requirements analyst. An ideal method would allow for the transparent capture of the information discussed at the meetings and the efficient organization of this information.

Rather than attempting to perform both scoping activities and the gathering of needs and objectives from users and customers simultaneously, some methods define organizational and context analysis as an explicit first step in elicitation, to be followed by other information gathering activities. These methods, such as the one presented in [Mittermeir 90], emphasize the importance of an objectives analysis for defining the organization's objectives, constraints against full achievement of those objectives, and their influences and interactions. The method described in [Mittermeir 90] does not contribute any additional technique to Table 4-1 and hence it will not be discussed further here. It does prescribe a different way in which a technique such as interviews can be used to first perform objectives analysis before other elicitation activities.

The ORDIT methodology, the product of an ESPRIT II project, also emphasizes the definition of organizational requirements [Dobson 91], [Dobson 92]. The ORDIT ("Organisational Requirements Definition for Information Technology") methodology explicitly recognizes that users tend to work in a collaborative or cooperative way in order to achieve an overall objective, and the aim of the ORDIT methodology is to "produce IT systems which match not only the organisational and functional needs of the individual end user, but also those of groups of users and their associated usability and acceptability requirements" [Dobson 91, p. 334].

4.2 Requirements Expression and Analysis

Elicitation is concerned with gathering information from various stakeholders in order to derive the requirements for a system. This collected information needs to be represented in some way, and ideally the gathered statements are expressed "in a notation which elucidates their implications, prompts further questions, correlates different aspects, and facilitates detailed analysis" [Southwell 87, p. 195].

Many current elicitation approaches represent the requirements from different viewpoints, in order to promote understanding and the gathering of information from the many communities involved in elicitation. This viewpoint approach is exemplified by CORE, which is discussed further in Appendix A.4.

CORE, an acronym for COntrolled Requirements Expression, and the Analyst, a CORE support environment [Stephens 85], represented the "state of the art" in requirements analysis as of 1988 [Finkelstein 88, p. 186]. GEC Plessey Telecommunications, U.K. (GPT), report their experiences with CORE in [Milsom 89], and summarize its benefits as follows:

- CORE provides a framework for analysis
- it expresses requirements in a structured diagrammatic notation
  - the notation fosters communication
  - it is less ambiguous than natural language
- CORE identifies design constraints at an early stage
- CORE supports early verification because both the target system and its environment are modeled

The limited tool support for CORE is noted as an inhibitor for its use on medium and large development projects. GPT summarizes their assessment as "CORE is a powerful method for gathering requirements but it lacks a standard notation and effective tool support" [Milsom 89, p. 138].

To strengthen their requirements engineering process, GPT uses CORE in conjunction with Specification and Description Language (SDL), which is "a well-defined notation which can be used to describe both requirements and design, and for which good tool support is emerging" [Milsom 89, p. 138]. Using SDL in combination with CORE provides enhanced communicability, since SDL is a powerful and compact notation which is more familiar to engineers and customers than CORE. SDL also enhances traceability between requirements and design [Milsom 89].

CORE is one of the few truly prescriptive methods available for guiding requirements elicitation [Kramer 88]. However, CORE does not express timing behavior very well, and needs to better support more complex data descriptions [Kramer 87]. Also, CORE, like other top-down methods, does not support reuse well [Finkelstein 88, p. 187]:

> *...the underlying philosophy of methods like CORE, which proceed in a 'top-down' fashion from the identification of viewpoints, agents or the like, actively militates against re-use, which is inherently 'bottom-up.'*

In CORE's combined viewpoint modeling stage, "only a subset of all possible transaction sequences can be considered", but despite this limitation it is noted in the same source that "this stage is a meaningful verification of the product specification" [Milsom 89, p. 139]. Therefore support for verification is provided, although it is incomplete and not enforced. Finally, despite CORE's utilization of viewpoints, including user viewpoints, other requirements engineering researchers argue that more is needed and that "there is still a lack of an overall methodology for incorporating the user perspective into requirements specification" [Macaulay 90, p. 94].

Other requirements elicitation process models described in the literature make use of views to promote understanding and structure information gathering activities. One such process model is the Planning and Design Methodology (PDM), outlined in [Mays 85]. PDM is guided by the following principles:

- Understand the rationale of the requirement.
- Verify the rationale and the proposed solution with the customer.
- Define the operational environment.
- Prioritize and establish business justification.
- Emphasize usability as well as function.

The first two principles address the importance of understanding and verifying the requirements of various stakeholders as well as addressing the "why" underlying those requirements. A technique such as gIBIS or a features model can be useful for capturing this rationale. The third principle emphasizes contextual factors, while the principle concerning prioritization is one way of dealing with risk and requirements volatility. One way of achieving this prioritization is through the use of Quality Function Deployment, in the manner described in Appendix A.5. The final principle on usability highlights the needs of one particular stakeholder group: the end user community.

PDM consists of requirements collection, problem analysis, solution definition, and system design. The three activities of synthesis, analysis, and communication occur within each phase. Synthesis is the construction of a work product from a number of initially unrelated sources of information. Analysis is the examination and verification of a given work product for consistency and completeness. Communication concerns the review and approval of the resulting documents by technical and nontechnical personnel. As with CORE, the concept of viewpoints is used by PDM to present a structure whose content is relevant to various types of reviewers.

An open issue with the use of multiple views is how to best communicate those views back to their authors. For example, GPT used a single representation (SDL) to communicate the requirements, as did the elicitation approach presented by Jokela and Lindberg [Jokela 90], which made use of Statecharts for both the full requirements model and user oriented models.

Other methods tailor the representation of these viewpoints to the different views. For example, Deutsch presents a method where real-time systems are described by the requirements model, the operations-concept model, and the implementation model. These three models represent the viewpoints of the customer, the user, and the implementer, respectively. Deutsch notes "there is no single modeling method or language rich enough to represent all aspects of the system and still be understandable.... Multiple views are needed" [Deutsch 88, p. 40]. Understanding is enhanced through the use of multiple expressions, but integration and verification is hampered. An alternative approach is to use a single means of expression so that integration is easier, perhaps at the expense of understanding. There are potential problems with integrating multiple views, however [Cameron 86, p. 239]:

...while it is a very appealing idea to make multiple descriptions of the same system, in effect to be allowed to view the same system from different perspectives, the idea does not really become useful unless there is some way to put the different descriptions together.

A newer approach to requirements elicitation which incorporated the notion of viewpoints is the multiple viewed approach defined by Harry Delugach in his Ph.D. thesis [Delugach 91]. His research addressed Cameron's criticism of how to put the different views together: conceptual graph representations are used for expressing the views, with structural analysis then performed on the conceptual graphs to associate views and find counterparts.

This approach has the advantage that views can still be captured using familiar techniques such as entity relationship diagrams, data flow diagrams, and state transition diagrams, with the conversion from these representations to extended conceptual graphs being done automatically. The meaning of concepts is no longer based on their names, but rather on their conceptual graph representations, and the multiple views promote both understanding and consistency checking.

However, a major criticism of the approach given the volatility of requirements is that "it is not clear how analysis could be incrementally performed, i.e., just the affected parts reanalyzed" [Delugach 91, p. 59]. Other problems concern how to get a view expressed initially, since:

- The participant does not yet know what is possible or reasonable.
- Many actual requirements are pervaded by implicit assumptions.
- Different people have different ideas of what they want in a system.

Finally, even with the views expressed finding the overlaps between views is a large polynomial problem. Multiple views are assumed to be at the same level of detail, nonfunctional requirements are not considered well, there is no consideration for rapid prototyping and traceability, and there is no support for incremental change. Also, the author admits that "unfortunately, we do not have the time or resources to properly measure multiple views' cost effectiveness on an actual project" [Delugach 91, p. 91].

Another elicitation approach to improving understanding is based not on views but rather on models. Models are considered as alternative representations of the requirements, such as data flow diagrams. The use of models for requirements elicitation has been noted in [Cameron 86], [Benyon 87], [Leite 87], [Deutsch 88], [Colbert 89], [Goodrich 90] and numerous other sources. For example, Jokela and Lindberg utilize user-oriented models to improve the communication between the users and the analyst, which also stresses the evolutionary nature of model building [Jokela 90, p. 294]:

In this paper, our discussion was based on a top-down approach, from the full model to the user models. Bottom-up approach is also applicable. In some applications the user interface and user operations are utmost important. We can first specify the system operation from the user viewpoints, and thereafter expand the model to a full one. In a typical realistic situation the analysis process is iterative where different user models and the full model are developed intertwined and iteratively.

Many elicitation techniques emphasize the use of a graphical notation to make the requirements models easy to read, thereby allowing these models to form "a good basis for communication between the analyst and the users" [Jokela 90, p. 289]. Benyon and Skidmore claim that "the visual attractiveness of the DFD [data flow diagram] makes it more effective than verbs" of natural language representations [Benyon 87, p. 3].

This claim was not supported by a study [Nosek 88] which found that the choice of representation, be it graphical or textual, had no effect on the level of user understanding of system requirements. The methods tested (HIPO, system flowcharts, DFD, narrative text, and Warnier-Orr diagrams) all were equally effective statistically in achieving user validation of the requirements document. This suggests that the actual form of the model is not as important as the semantic content of the model. The content, and not solely the form, of the model determines its value. This point is emphasized repeatedly by Leite [Leite 87] and supported by the following quote of John von Neumann [Weinberg 88, p. 157]: "There's no sense being precise about something when you don't even know what you are talking about."

Other research work on new approaches to requirements elicitation concentrates on precision and the use of formal models for representing requirements. These formally based techniques do not scale up well to practical industrial software development in large part due to a lack of basic tool support [Finkelstein 88, p. 186]. Another reason offered to explain why these techniques do not scale up well is "because little attention has been paid to the method of guiding and organising the activity by which a formal specification is obtained from an informal application concept" [Finkelstein 86, p. 236].

One inhibitor to using models during requirements elicitation is the difficulty of developing such models: "in practice the significant abstraction of the problem domain may be hard to find and the skill of making such abstractions is hard to teach" [Cameron 89, p. 2/1]. The difficulty in function-based structured analysis methods is in identifying appropriate functions and data flows. Similarly, "identifying the objects in a problem is the hardest part of this or any object-oriented method" [Colbert 89, p. 414].

Elicitation techniques such as CORE, Delugach's multiple views, or the use of various models can assist in requirements expression and analysis. A criticism of most of these elicitation techniques, however, is that they do not scale up well. They do not effectively handle the elicitation of requirements for a large problem area or affecting a large number of people. Hierarchical decomposition of the problem area into manageable parts is one way requirements analysts can address this issue [Dubois 88]. Such "modularization" is important during the elic-

itation of requirements [Macaulay 90]. Most methods for analysis and design have at an early stage some way of decomposing the problem into manageable areas [Kramer 88]. There may be many criteria that suggest one decomposition over another, e.g., organization of the user's institution, reusability, or extensibility [Bailin 89]. Decomposition is often explicitly noted as a phase of the elicitation process [Zeroual 89].

Top-down decomposition can be criticized for placing too much importance on early decision-making [Cameron 86, p. 239]:

> *Each decision about the decomposition of a subsystem depends on the decisions that have lead to that particular subsystem. This hierarchical decision structure makes the early decisions very critical. A bad early decision may not be discovered until very late. The designer must exercise tremendous foresight to make good decompositions.*

Returning to the technique of multiple views, understanding a large problem domain may not just be a matter of decomposition but of composition as well, composition of multiple views. The multiple view approach expressed in Delugach's thesis [Delugach 91] postulates that effective requirements development depends upon the participation of everyone affected by a proposed system. Each participant involved in the requirements development process has a different view of the target system, and describing any participant's view of the system or environment will tend to improve the overall understanding of that system. The final requirements are composed from the multiple participants' views.

As a problem becomes more complex and addresses a broader scope, it becomes increasingly unlikely that any individual will be able to understand the problem well enough to explain and develop comprehensive solutions and refine the problem formulation. Rather, only through a composition of multiple people's views of a problem will the requirements be able to adequately be elicited and defined. A compositional approach seems to be a necessary, albeit not sufficient, technique addressing problems of understanding in elicitation scenarios where the system is large and complex. With compositional approaches, however, problems of understanding still result from difficulties in integrating the multiple views of the system, and in having these views expressed consistently and in such a way that they are understood by all the elicitation partners.

4.3 Validation

One often cited technique for dealing with requirements volatility is to validate early and often that the information gathered so far and the representation of that information is consistent with the elicitation communities' needs and expectations. This repeated validation is part of an incremental approach to elicitation, a concept introduced in Section 3.3. Requirements elicitation should also be interleaved with subsequent activities such as the design and early prototyping of critical components [Andriole 90, p. 6]:

Consequently, systems analysts developed a new design perspective, one that assumes that requirements cannot be captured the first time through and that several iterations may be necessary to define requirements accurately. This new perspective is anchored in the value of ***prototyping.***

Prototyping of core functionality can be used to develop early executable versions of the requirements model. Indeed, prototyping has been cited as a cure for the problems of understanding during requirements elicitation in many sources [Jordan 89], [Andriole 90]. However, prototyping is not the silver bullet [Dubois 88, p. 394]: "it may be costly to develop a prototype, which is not the ultimate solution anyway; users are assumed to accept the prototype when they actually only accept its behavior in the cases they have tried."

Quality Function Deployment (QFD) is another useful technique for validation. QFD allows the "Voice of the Customer" to be captured, and then the proposed requirements of the system to be validated based on whether or not they reflect these expressed customer needs [Schubert 89]. As outlined in Appendix A.5, QFD helps to identify user requirements that have not been addressed by the developer, as well as developer-proposed features that do not support any requirements. In addition to highlighting such omissions, QFD also documents requirements that are highly rated by the user and receive little attention by the developer-proposed features.

5 An Elicitation Methodology Framework

Many requirements problems are due to poor requirements elicitation, including the resulting requirements being ambiguous, incomplete, not verifiable, inconsistent, irrelevant, and not correct because they do not reflect the stakeholders' needs and objectives. These problems stem from issues of scope, communication, and requirements volatility. There are elicitation techniques which address some of these issues, as discussed in the previous section and the appendices. However, no technique is comprehensive enough to adequately cover all of these issues in detail. Rather than advocating one technique over the others, a better approach to requirements elicitation is to synthesize the various methods and techniques into a methodology, which then can be instantiated based upon a target system's attributes.

This section will present a process model which focuses attention on the problem areas of requirements elicitation. It then will outline a strategy for requirements elicitation which improves upon past elicitation techniques while incorporating their advantages. Based on this strategy, a tailorable elicitation methodology will be presented, along with a discussion of the difficulties of evaluating such methodologies.

5.1 A Requirements Elicitation Process Model

Recall that Section 2 established the context for requirements elicitation within the system development process. Elicitation deals with fact-finding, information gathering, and integration in order to obtain a set of requirements which describe a number of possible solutions. The remainder of the requirements engineering process is concerned with validating that these requirements satisfy the goals of all parties affected by the system being built, and with communicating these requirements to developers via a specification. This section shall propose a process model for requirements elicitation which takes into account the problem areas discussed earlier.

The requirements analyst is the party responsible for the capture of system requirements from the user community and the communication of these requirements to the developer community. In addition to these tasks, the requirements analyst must also make sure that the needs of other affected communities are reflected in the gathered requirements, and that a proper understanding of these requirements is communicated to all affected parties.

The process model outlined here recognizes the importance of this communication between different stakeholders. It is in agreement with the paradigm underlying the Issue-Based Information Systems (IBIS) method [Begeman 88, p. 255]:

> *The IBIS method...is based on the principle that the design process for complex problems is fundamentally a conversation among the stakeholders (i.e., designers, customers, and implementors) in which they pool their respective expertise and viewpoints to resolve design issues.*

Recognizing the importance of communication is not enough, though. The backgrounds and motivations of the elicitation participants are often very different, and the process model consists of two sets of activities to address this diversity. One set of activities is user-oriented, while the other is developer-oriented.

The two sets of activities are performed in parallel, and can be grouped into tasks associated with fact-finding, requirements gathering, evaluation and rationalization, prioritization, and integration. These task groups may be executed iteratively, as illustrated in Figure 5-1. This process model is similar to the system requirements capture model outlined by Mittermeir and others [Mittermeir 90], but in that model prioritization occurs before rationalization.

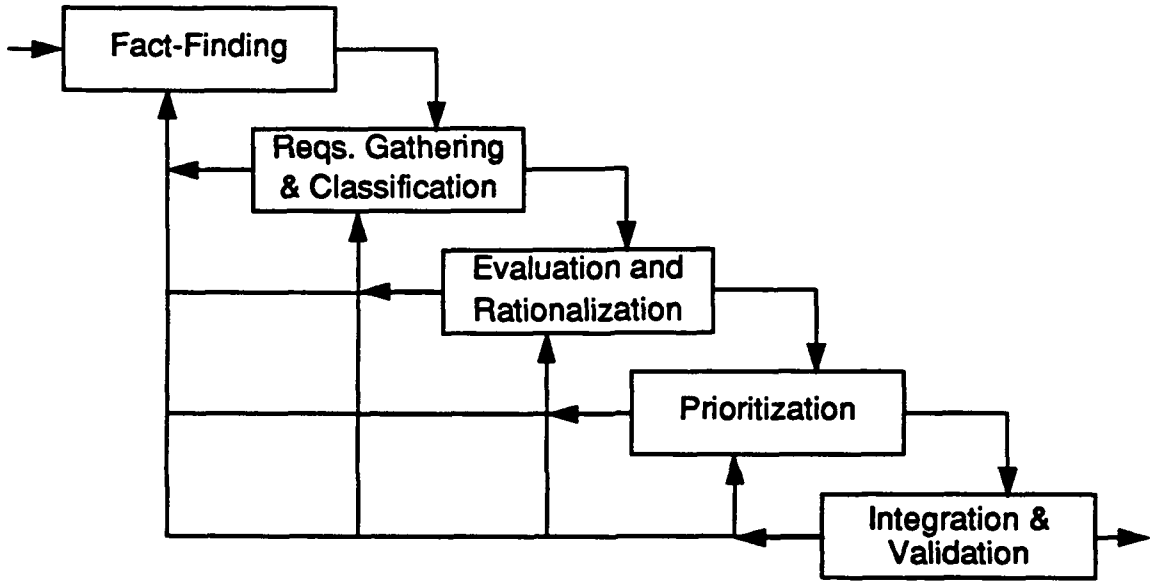

Figure 5-1 Proposed Requirements Elicitation Process Model

This elicitation process model is first executed during the concept exploration phase of system development, which is initiated after the creation of a mission needs statement. Following this phase, the first level of detail in the requirements specification is achieved. During the subsequent demonstration and validation phase, these specifications are validated, the unclear requirements clarified, the unknown requirements identified, and the existing ones refined as necessary. Based on communication mechanisms (such as prototyping) employed during this phase, these elicitation steps are then cycled through again beginning with "requirements gathering" to detail and improve the requirements document. Recall from Section 3.3 that not all of the requirements for a system are typically known immediately. This implies that the demonstration and validation phase is entered with incomplete requirements, and therefore that these elicitation process steps are returned to after the first pass through the concept exploration phase.

User-Oriented Tasks	Developer-Oriented Tasks
Fact-Finding Identify relevant parties across multiple levels. Determine operational and problem context, including definition of operational modes, goals, and mission scenarios as appropriate. Identify similar systems. Perform context analysis.	**Fact-Finding** Identify domain experts (both application area and development experts). Identify domain and architectural models. Conduct technological surveys, for later feasibility studies and risk assessment. Assess cost/implementation constraints imposed by the sponsor.
Requirements Gathering and Classification Get wish list for each party across multiple levels.	**Requirements Gathering and Classification** Classify wish lists according to functional, nonfunctional, environment, and design constraints; and also according to partitions defined by domain models and the development paradigm (e.g., top-down functional decomposition or object-oriented).
Rationalization and Evaluation Perform abstraction to answer questions of the form "Why do you need X?"; this in effect moves from statements of "how" to statements of "what". Capture rationale to support future requirements evolution.	**Rationalization and Evaluation** Perform risk assessment, addressing technical, cost, and schedule concerns (this includes cost/benefit filtering and feasibility analysis based on technology availability).
Prioritization Determine criticality, i.e., the critical functions for the mission.	**Prioritization** Prioritize requirements based on cost and dependency. Study how the system can be incremented, and identify appropriate architectural models which support incremental development.
Integration & Validation Address completeness by filling in as many "to be determined" issues as possible. Validate that requirements are in agreement with originally stated goals. Obtain authorization/verification to move to the next step of development, e.g., the demonstration and validation phase.	**Integration & Validation** Resolve conflicts (consistency checking).

Table 5-1: Tasks Comprising the Elicitation Process Model

The tasks which compose this process model are listed in Table 5-1. With regard to the user community, fact-finding begins with identifying the relevant parties at multiple levels within the community, e.g., from a high-level commander for a strategic long term perspective to an end user for the immediate perspective. The operational context and problem context are defined,

perhaps through goal trees and mission statements, which help with the later filtering of the requirements. This includes an objectives analysis, which studies the user organization's objectives, constraints against full achievement of the objectives, and their influences and interactions [Mittermeir 90]. Context analysis and the determination of operational modes and mission scenarios completes the user-oriented task fact-finding activities. The developer oriented fact-finding tasks are performed in parallel. These fact-finding tasks are important [Berlin 89, p. 93]:

> *Studying user needs is a first step to any solution, along with gaining an understanding of available technologies and existing tools. These two tasks interact. Without an understanding of technologies one may aim for the impossible, and without an understanding of needs, one may solve the wrong problem.*

The communication between the user-oriented and developer-oriented activities is cyclical, and enhanced via modeling. The communication enhancement is desirable. The representation of the requirements should promote understanding while allowing for inevitable change, and hence this representation should be introduced as early into the requirements engineering process as possible while still maintaining the desirable characteristics of modifiability (extensibility and evolvability), readability, and analyzability.

This process model will be discussed further in Section 5.4, along with techniques that can be used to achieve the tasks in the model.

5.2 Methodology over Method

The degrees to which requirements will be influenced by contextual factors, require communication between different communities, involve large quantities of data, and change over time are dependent on the target system, i.e., the proposed system to be developed. To address this dependency, the requirements elicitation approach to solving the problems stated earlier will be a methodology, as opposed to a single method. The difference between these terms is stated eloquently as follows [Checkland 89b, p. 101]:

> *It is the essence of a **methodology**—as opposed to a method, or technique—that it offers a set of guidelines or principles which in any specific instance can be tailored both to the characteristics of the situation in which it is to be applied and to the people using the approach...Such is the variety of human problem situations that no would-be problem solving approach could be reduced to a standard formula and still manage to engage with the richness of particular situations.*

Target systems have different organizational and environmental contexts. For example, one target system might be a complete stand-alone system with very few environmental constraints, while another might be a very small component of a larger existing system, with lots of interfaces to that larger system and constraints imposed by the existence of that larger system.

Different target systems also involve different elicitation communities. One target system might be developed by the users themselves because they have both the domain experience and the development experience. Another may have disjoint user and developer communities.

Target systems will have varying amounts of information necessary for elicitation, e.g., the users who are also developers working on a small stand-alone system will need less data than the separate developer community building a critical component of a larger system for many different user communities.

Finally, a target system will have different degrees of stability, with the replicating of a long-standing manual task being more stable than the development of a task which does not have a parallel in current practice.

The approach to requirements elicitation presented here shall thus be problem-specific. Attributes of the target system, including organizational constraints such as the backgrounds of the involved communities, the number of environmental constraints, project constraints such as cost and time limitations, and the potential for evolution (as determined by the domain's maturity, the availability of standards, and other factors), will serve to instantiate the method to use for elicitation for that particular target system. The instantiated method is derived from a general elicitation methodology.

This problem-specific approach is endorsed in [Andriole 90, p. 9]:

> *...the problems the prospective system are intended to solve should determine life cycle assumptions. Designers that begin a priori with a method will often fail, if only because they may end up matching the wrong life cycle assumptions with the wrong problems.*

Andriole notes that analytical problem-solving requirements are best elicited through iterative techniques, while problems with an absence of analytical requirements might well be modeled via conventional, sequential methods.

Determining the elicitation approach to use based on the attributes of the problem is also recommended in [Benyon 87]. In this paper, Benyon and Skidmore define five systems analysis approaches:

- soft systems approach: characterized by Checkland's Soft Systems Methodology (SSM), which is discussed briefly in Appendix A.6; emphasizes the subjectivity of systems analysis, i.e., there are a variety of legitimate views of a problem situation, and the importance of an iterative approach to the activity, e.g., the use of conceptual models to contrast the desired and actual states of a system
- structured systems analysis and design: characterized by DeMarco's data flow diagrams; emphasizes data, that the systems represent transformations, and the concept of hierarchy in developing models, e.g., start with context diagram and work toward functional primitives

- traditional approach: appropriate for batch systems but not flexible enough to cover the range of modern systems; focuses on functional analysis, fact-finding, and flow of control
- data-centered approach: looks at a static representation of the information content of a system; it is independent of technology and hence allows a variety of implementations, but "its emphasis on data, coupled with the apparent rigour of its techniques, can lead to the development of neat technical systems that ignore or contradict the political reality" [Benyon 87, p. 4]
- participative approach: characterized by Mumford's socio-technical system design [Mumford 81] and the use of prototyping; emphasizes understanding by users of what they want and what is possible, an approach that responds well to complexity and uncertainty

Rather than identify a particular approach which is the most valuable, the authors recommend the following [Benyon 87, p. 5]:

> *We feel that the five methods are essentially complementary. Discussions about which is best seem rather fruitless, because the success of an approach is dependent upon so many external variables. It seems more useful to see the five approaches as comprising tools available to the analyst, who then chooses the correct tool or set of tools for a particular set of constraints and circumstances.*

A number of alternative methods for requirements elicitation are outlined in broad terms in a paper by Stair and LaMothe [Stair 91], followed by a list of contingencies to determine which approach to follow. Approaches for determining information requirements include:

- determining directly
- deriving from existing systems
- normative analysis (develop a generic model and then successively modify the model to meet user needs)
- strategy set transformation
- critical success factors
- key indicator analysis
- prototyping
- scenarios
- information needs analysis

It is noted that selecting the best requirements method from this list "can be as difficult as determining information requirements" [Stair 91, p. 35]. The method to be used should be contingent on the specific circumstances of the organization, including user experience, the level of internal and external uncertainty, a clear statement of goals, overall orientation, and the lev-

el of assistance. For example, with high user experience and low uncertainty, determining directly is the recommended approach, while with low user experience prototyping is recommended.

The recommendations presented by Stair and LaMothe [Stair 91] begin to address the difficult task of instantiating an elicitation method depending on organizational factors, but are often too simplistic in that for many cases, a single organizational factor is the only one used to recommend a given method. For example, the scenario technique is recommended if and only if there is high external uncertainty.

It may be useful to define "method" and conclude this subsection with a contrasting of "method" to the "methodology" definition given above. Referring once again to the requirements engineering literature, Kramer and others define a method as consisting of a grammar of steps and principles for applying them rather than just a collection of notations [Kramer 88, p. 91]. Likewise, Mullery notes the presence of application rules in a method, defining a method as "a collection of models for specification and a set of rules for their use" [Mullery 89, p. 1/1]. Any single model is an incomplete representation of a real entity, so several models may be necessary to illustrate or analyze different aspects of the entity. Examples of low level models for expressing requirements are state transition, procedural decomposition, process/data flow networks, organizational structure and decision support. Finkelstein and Potts note the constraints that a method's application rules place on the requirements engineering process [Finkelstein 86, p. 237]: "A method necessarily *constrains* the practitioner's freedom."

The strategy expressed here is based on the notion that the selection of a method, with its inherent constraints and sets of application rules, cannot be done independently from a consideration of the target system. Rather, an elicitation approach ideally is instantiated from a methodology in order to best address the particular target system's organizational constraints, architectural restrictions, and involved audience's backgrounds.

5.3 Integration of Techniques

The problems of Section 3 are best addressed through the development of a tailorable elicitation methodology in agreement with the process model described above, along with guidelines on how to fit the methodology to different projects. The methodology must address the organizational, architectural, stakeholder, and problem-specific issues identified earlier.

Current work on requirements analysis has focused on the structure and notations for specification, without providing any guidance as to how the specification is to be elicited from the user in the first place [Kramer 88, p. 86]. The elicitation methodology should eventually be prescriptive in nature in order to provide this guidance. However, this methodology is only being *introduced* in this section. It is important to note that guidelines for tailoring the methodology to specific problems will most likely be developed, validated, and refined iteratively. As the methodology matures and more problem areas are addressed, the framework will grow as well, so this section should not be treated as the definitive, comprehensive solution to requirements

elicitation. Rather, it should be seen as a first step towards integrating partially successful past elicitation techniques into a more useful methodology based on the process model presented in Section 5.1.

The proposed elicitation methodology will be composed of the following:

- fact-finding: this includes the following:
  - an examination of the organization into which the target system will be placed
  - high level statements of the target system's mission or role
  - determination of any constraints on the architecture
  - determination of the existence of similar systems

  Typically the system to be developed is a component of some larger system. There will likely be many people affected by its development as well. The socio-technical approach to fact-finding espoused here is similar to the ORDIT methodology where the system is "viewed as a whole by placing it within the broad operational environment with the user as an integral part of the system" [Dobson 92]. Organizational analysis addresses *who* is knowledgeable about the target system and who will be affected by it. Architectural considerations, as well as project constraints such as limits on cost, help to answer *how* something is to be built.
- requirements gathering: capture information through the use of multidisciplinary views. Such views express *what* is to be built.
- evaluation and rationalization: expose inconsistencies in the gathered requirements. Determine *why* something has been expressed as a requirement.
- prioritization and planning: determine the relative importance of the requirements, i.e., answering *when* requirements should be addressed in relation to each other.
- integration and validation: bring together the information collected from the previous steps into a set of requirements, and validate that these requirements are in agreement with the goals originally extracted during fact-finding.

5.3.1 Fact-Finding

Referring to Table 5-1, the very first step in requirements elicitation involves determining what is the problem to be addressed, and who needs to be involved in this decision-making as well as who will be affected by the problem's formulation and eventual solution. The output from this activity includes:

- a statement of the problem context
- the overall objectives of the target system
- boundaries and interfaces for the target system

In well-understood problem domains, much of this information may already exist in some form and be readily available. In other cases the definition of the problem context may be the most difficult elicitation activity and may need multiple passes in concert with validation activities before it is done correctly.

Tasks at this stage include an analysis of operational, problem, and organizational context, identification of similar systems, and the assessment of cost and implementation constraints imposed by the customer. The problem scoping activities involved in these tasks are vague and open to interpretation. It is best to have all the affected parties participate in this stage, including users, developers, and customers. This results in shared ownership among these communities of the context analysis and problem formulation process, which will impact positively on the volatility of the requirements. If everyone involved agrees up front to the scoping, there is a better chance that the "right problem" will be addressed and that major changes in the requirements will not occur later in the development process.

An effective approach to achieving this cross-disciplinary communication for fact-finding is the use of a group process technique, such as JAD. All the affected parties should be represented in the group which will perform these early fact-finding tasks. This promotes shared ownership, rapid early problem formulation, and an aligned perspective and understanding between the elicitation communities of the problem to be solved and the scope of the subsequent requirements.

Within a structured meeting technique like JAD, other techniques can be used to promote communication between the individuals from potentially many disciplines. SSM's "CATWOE" information gathering model, discussed in Appendix A.6, can be used to identify the essential objectives of the problem situation. IBIS, discussed in Appendix A.1, can be used to record the different issues, their associated positions, and arguments for and against those positions which are raised during the group discussion. This record can then be used to support the consensus generation activities which are part of the JAD process. Such a record is also very valuable given that this fact-finding stage will likely be returned to iteratively during the requirements elicitation process.

Later passes through fact-finding may not necessitate a structured group meeting technique such as JAD to refine the requirements. It may be that subsequent passes through the elicitation stages outlined here only touch on fact-finding, such as when a new community of users is recognized as being affected by the development of the proposed system.

5.3.2 Requirements Gathering

Depending of course on the system being developed and the groups which will be affected, the requirements gathering stage is a combination of both compositional and decompositional approaches. In early problem formulation stages, it is important to gather as much information as possible from users, developers, and customers. Some of this information may come from the group development techniques employed during fact-finding, such as JAD. More information can be gathered through the use of interviews directly with end users and other affected

parties. Questionnaires, observations, and simulation environments are other techniques that can be utilized to get information from different individuals and groups [Andriole 90]. The output from this activity includes:

- customer and user oriented objectives
- customer and user oriented needs and requirements

These needs and requirements are verified against the overall objectives of the target system expressed during fact-finding.

When there are many customers or users contributing requirements to the analyst via techniques such as interviews, a number of sets of needs and requirements, or views, will be collected. It is very difficult for the analyst to identify and resolve inconsistencies between these different views if there is no additional structure to the information. One way to provide this structure and organize information from multiple individuals is through the use of the CORE method for requirements extraction, as discussed in Appendix A.4. Eventually, these multiple views are then composed into the requirements for the system.

The views are better understood if they can be structured into manageable pieces. This is especially true given that the elicitation process will be incremental, to deal with inevitable changes in requirements. If we return to monolithic views of the complete system, it will be very difficult to both comprehend such a large view and also to find portions of that view which may be affected by an incremental change to the requirements. Thus, there must also be a decompositional process associated with requirements gathering, where the views can be broken down into meaningful components.

Ideally, there are applicable domain and technological models for such decomposition. For example, with the domain modeling approach outlined in Appendix A.2, a number of models can be used such as the features model and entity-relationship model to better communicate to the elicitation parties the information expressed in that view. CORE emphasizes the use of data flow diagrams for such decomposition of views. The representation used for such decomposition is very dependent on the application domain, e.g., the data flow representations of CORE are noted as failing to adequately capture formalism which may be necessary in requirements for embedded systems [Finkelstein 86].

5.3.3 Evaluation and Rationalization

A risk assessment should be performed to address technical, cost, and schedule concerns. In addition, the rationale behind the information gathered in previous stages should be examined to determine whether the true requirements are hidden in this rationale instead of being explicitly expressed. This rationale and risk assessment are the two main outputs from this activity.

IBIS is a well-suited technique for capturing the rationale in the issues-positions-arguments framework during group development techniques and interviewing. Structured models of the domain and the technology are extremely useful. For example, with the features model described in Appendix A.2, issues are identified where alternative selections can be made, and

decisions describe these alternatives and the rationale associated with these alternative selections. Given a view's decisions on these issues, we automatically gain an understanding of the rationale associated with the decisions via the features model.

Once the rationale has been collected and examined, inconsistencies can ideally be found and better choices on decision points or issues made to both resolve these inconsistencies and address the needs reflected in the rationale. In addition, this rationale is extremely useful in later passes through the elicitation methodology as documentation on why particular choices were made. If incremental changes to the requirements are to be made, these changes can be checked to see if they are in agreement with the rationale underlying the rest of the existing requirements.

5.3.4 Prioritization

Incremental development, of both the system and the requirements, is stressed in the process model outlined in an earlier section. If requirements are prioritized, then high priority needs can be addressed first, and the subsequent requirements changes defined and reexamined, before low priority needs (which could change as well) are implemented. This can result in cost and time savings when processing the inevitable requirements changes during system development.

The requirements must be prioritized based on cost, dependency, and user needs. Architectural models can help with dependency relationships and determinations on how the system can be incrementally developed. QFD is an ideal technique for determining critical system features and prioritizing user needs.

5.3.5 Integration and Validation

An important advantage of group development techniques such as JAD is that they promote shared ownership of the developed requirements, with an improved definition of scope as well as reduced chance for future requirements changes. These techniques stress that integration of multiple views should occur as much as possible through the involvement of all the affected communities, so that this shared ownership is not lost. If integration is done by a single elicitation community such as the developers following fact-finding, requirements gathering, and the other earlier stages, then the integration will be viewed as the developers' interpretation of the requirements, and may be criticized as not incorporating other important interpretations as well.

There are a number of ways of handling this problem. One is to repeat a group development technique later in the elicitation process, so that group ownership of the integration may again take place. Since senior level managers are often involved in such group development, it may be impossible to get such commitment both during fact-finding when their contributions are critical as well as at the end of the elicitation process. Such involvement by senior personnel at both the start and end of conceptual development may be rejected in favor of involvement

of these people at the start of conceptual development, and then later during demonstration and validation, with corrective feedback loops going from demonstration and validation and later phases back to conceptual development.

The integration tasks might be performed primarily by the systems analyst, and the results of the elicitation process communicated to the other involved communities through various strategies such as prototyping. These communications can be viewed as lying outside the realm of concept exploration and requirements elicitation and instead being the first activities of demonstration and validation. This validation of the requirements by all affected parties ensures that their concerns are met. Subsequent passes through the elicitation methodology outlined here address the requirements deficiencies, inconsistencies, and other problems found during the demonstration and validation steps.

In cases where the requirements analyst is responsible for integrating the information from the previous stages into a set of requirements meeting all the desirable attributes outlined in Section 2, there are a few techniques which can be used. For example, the existence of domain models and architectural models provide "maps" for how to organize information in a requirements document, and how to present this information to the different elicitation communities.

When multiple views are used, these views can be integrated together through the conceptual graph techniques described in Delugach's Ph.D. thesis [Delugach 91]. This is a nice automated approach to viewpoint integration, but does not support incremental requirements development very easily and also places some assumptions on the different views, such as that they are descriptions of the problem given at the same level of detail.

QFD can also be used for integration and validation by relating the user and developer views of the system, as discussed in Section 4.3. The integration of user and developer views may reveal shortcomings which necessitate that earlier stages of the elicitation methodology be revisited.

5.3.6 Methodology Summary

This methodology may be criticized as being too abstract, but these concerns are offset by the benefits of being inherently flexible, which are noted in Section 5.2. Both organizational and individual constraints are taken into consideration by the methodology, in agreement with [Macaulay 90, p. 94]:

> *The new approach would need to adopt a symbiotic approach which recognises that individuals must satisfy organisational needs and equally organisations must satisfy individuals' needs, and all of these needs must be satisfied by the product.*

Difficulties in applying this methodology due to its generality can be overcome by specializing it according to a given elicitation scenario and the characteristics of the affected parties. For example, the development of a novel system affecting a number of user and client communities may very well focus a great deal of effort on fact-finding, with additional techniques to aid

in this stage of elicitation as well as more guidelines for conducting fact-finding sessions. The development of a system revision involving the same users and clients as earlier may focus less on fact-finding and instead have additional techniques and guidelines for prioritization and integration.

The methodology outlined here does not discuss how these specializations to meet different scenarios and elicitation communities should be defined. Rather, it provides a general framework with the assumption that requirements elicitors will tailor it accordingly. This philosophy was expressed earlier, and is in agreement with the philosophy behind SSM, discussed in Appendix A.6.

The focus of the methodology is on communication issues and stakeholder issues, as opposed to issues of notation. Too much early emphasis on notation has been pointed out as a problem with elicitation in Section 4.2. This advice is repeated in an argument against the early use of formal methods [Macaulay 90, p. 100]:

> *While there is a role for formal methods in the requirements specification process, it could be argued that the initial stages of requirements capture are necessarily vague, ambiguous and incomplete, as they are the subject of much negotiation between individuals. Formal methods can be most successfully applied after the initial user requirements have been agreed.*

5.4 Evaluation Criteria

Ideally, following the definition of a new elicitation methodology an evaluation study could be run to determine the benefits provided by that new methodology over current practice. However, there are many difficulties associated with the evaluation of requirements engineering methodologies, a few of which will be discussed here.

Two evaluation criteria for requirements capture methodologies are:

1. effectiveness, the achievement of the highest valued goals; and
2. efficiency, achieving goals without consuming more resources than necessary.

Unfortunately, these criteria can be in conflict [Fowler 90], so a methodology which excels on effectiveness might be very poor in efficiency and vice versa. The goals of a methodology need to be clearly stated in order to adequately judge how well that methodology achieves its goals and to determine whether a given methodology is successful.

Evaluation studies of development methodologies are plagued by the difficulty of controlling for developers' experience and level of creativity, and the unlikelihood of commercial developers employing parallel designs for the objective comparison of methodologies. Evaluation strategies designed to overcome control problems and yet maintain the necessary rigor include the following, which are presented along with the authors' criticisms [Fowler 90]:

- Develop a taxonomy of parameters which define a particular class of methodology. The major problem with this approach is that the methodologies do not all address the same parameters. Also, the evaluators generate the parameters from some notion of an ideal methodology, and in many cases the methodologies differ because "there is not or cannot be an idealised form" [Fowler 90, p. 192].
- Develop various "yardsticks" against which the features of different development methodologies can be compared. No value judgment is implied by this approach, but the weakness in this approach is that "yardsticks are generated empirically by analysing the content and approach of established methodologies to determine what factors have been considered useful in the past" [Fowler 90, p. 192]. The criteria should not only come from past behavior. Also, a number of comparable methodologies may not exist in a given area.
- Use case studies to compare usage of a methodology with its developers' claims. Fowler notes that "it is fundamental to this approach that the evaluators are well trained in using the methodology" [Fowler 90, p. 193]. The reliability of a methodology may be difficult to measure because the same case is likely not repeatable, and case studies rarely involve a comparative element.
- Survey the methodology users. This really addresses the assessment of a methodology's external validity (i.e., it is usable by requirements engineers), not internal validity, and it also may not easily separate methodological issues from other contextual factors.

The authors conclude that "in the final analysis a requirements capture methodology is of value only if the end product is considered to be better, in some specified way, than products developed by alternative means" [Fowler 90, p. 194].

The evaluation of methods is dependent upon a number of factors, including the following [Hackathorn 88, p. 209]:

- current state of the information technology employed
- level of integration between methods
- perceived suitability in specific situations
- organizational experience with the method
- skills and expertise required in using the methods
- commercial support, pricing, and training involved

Because of these various dependencies, it is difficult to generalize the potential benefits and constraints of an information engineering method, including a requirements elicitation method. As noted elsewhere [Winokur 90, p. 88], it is essential that complete training be provided for the methodology and for all supporting tools before the method is applied to a real system analysis for evaluation. Without proper training in the methodology and support tools, a method may be misused and hence the evaluation of that method may be flawed.

Goodrich and Olfman evaluated requirements analysis methods based on the factors of Requirements, Commitment, and Understanding [Goodrich 90]. Requirements focused on completeness, accuracy, and relevance. Commitment referred to communication, expectations, involvement, and level of control over specifying requirements. Understanding concerned predictions about the system and the degree of understanding of the task and process. They concluded that "we believe that all three factors must be present in varying degrees for analysis phase success" [Goodrich 90, p. 203].

Requirements analysis methods can also be evaluated from the viewpoint of the whole development process. Six criteria that can be used to judge the success of a requirements engineering methodology in this fashion are given as follows [Macaulay 90]:

- design philosophy, e.g., it should create modifiable design solutions and be iterative in nature
- fitting, i.e., the methodology should fit with the current design practice, e.g., by supporting established design concepts; one useful way to establish requirements is through prototyping
- user orientation, i.e., the methodology should be acceptable to designers and should support and encourage multidisciplinary views, supporting conflict resolution
- support for project management, e.g., cost estimation, planning, scheduling, and productivity increases brought about perhaps through the use of automated tools
- quality assurance, i.e., ensuring that the specified requirements are pursued through the development process and can be traced back
- modeling, with the model exhibiting characteristics of users' views, human readability, precision, specification completeness, and mapping to later phases

This section has pointed out some of the problems with evaluations of methodologies, as well as some of the evaluation strategies that have been employed in spite of these problems. Hard evidence to support the use of one methodology over another will always be difficult, if not impossible, to produce [Checkland 89b, p. 118]:

> *Since the same human situation cannot be investigated twice, methodology is undecidable: 'successes' might have been greater with some other approach, and 'failures' might be due to incompetence using the methodology rather than the methodology itself.*

6 Conclusions

Requirements elicitation is a collaborative decision-making activity involving users, developers, and customers. The elicitation approach is dependent not only on the diversity and experience levels of these cross-disciplinary sources of requirements, but also the diversity of the problem being formulated, which ranges from a fully understood system to a new, novel one. Any single approach to requirements elicitation will have varying success across different projects, since "not all problems yield to the same degree to the same approach" [Cameron 89, p. 2/1]. For example, the development of a system with a simple problem domain may not be affected significantly by the use or avoidance of the JSD model. It therefore has been proposed in this paper that the requirements elicitation approach to use for a given project be tailored to that project.

Benyon and Skidmore summarize the problems with using one specific elicitation strategy[1] [Benyon 87, p. 7]:

> *We feel that it is unlikely (if not impossible) that a single methodology could prescribe how to tackle the great variety of tasks and situations encountered by the systems analyst...The desire to produce 'one best way' is leading to elaborate and bureaucratic methodologies.*

As an alternative, the strategy presented in this paper advocates the synthesis of multiple techniques into an elicitation approach that can be instantiated to address the attributes of a given target system. These attributes include:

- social constraints, e.g., what groups are involved and what are their backgrounds
- architectural constraints, e.g., what must this system interface with
- organizational constraints, e.g., time, cost, and personnel factors
- problem-specific factors, e.g., domain maturity and the availability of standards

The elicitation methodology proposed in this paper will address the problems listed in earlier sections and focus on achieving the properties of a good specification, and this will be accomplished by integrating flexibility into the methodology, as noted in [Macaulay 90, p. 101]:

> *There is a danger in being 'technique-oriented,' so that the problem situation will be distorted to fit the technique. We need rather to be problem- and user-oriented and to allow the situation to distort the way in which the analysis is carried out. This orientation demands flexibility in the approach, and hence the emphasis here is on methodology and not on technique.*

---

[1] The authors use "methodology" in the way "method" was defined in Section 5.2, rather than how methodology was defined in that same section.

There are a number of elicitation techniques useful for addressing one or more of the problems discussed in Section 3. However, no single technique adequately addresses all of the elicitation problem areas. The methodology proposed in Section 5.3 integrates various elicitation techniques to incorporate their advantages, while comprehensively addressing the full range of elicitation issues.

Appendix A Notes on Selected Elicitation Methods and Techniques

During the course of this paper, a number of elicitation techniques, methods, and strategies were introduced and often summarized as to their good points and deficiencies. A few of those techniques and methods are discussed further in this appendix.

A.1 Notes on IBIS

IBIS, an acronym for Issue-Based Information System, is a structuring method which allows the rationale underlying requirements to be organized and tracked [Conklin 88], [Yakemovic 90]. The IBIS method is used to capture dialogue information by keeping track of the issues being discussed, the positions on these issues, and the arguments in support of or objecting to positions.

IBIS has a number of advantages. First, it is a technique which is easy to learn and use. An indented text system was used in an early implementation of a tool for IBIS, and it was well accepted because the users were already familiar with the components (personal computers and text editors) used for the tool [Yakemovic 90].

IBIS results in consistency in the quality of the notes being taken. Also, the "important" information tends to be captured because of the structure imposed on meetings by the IBIS method. Thus IBIS results in more productive meetings. For example, open issues are saved and returned to later, reducing incompleteness. As another example, when tangents are brought up, they are stopped sooner and the interrupted issue, position, or argument returned to more quickly. The issue-position-argument framework helps focus thinking on the difficult, critical parts of the problem, and improve detection of incomplete or inconsistent thinking. Also, it tends to make assumptions and definitions more explicit sooner.

IBIS provides a cushion for the human resources problem mentioned in the SEI requirements analysis workshop [SEI 91]. If a person who has been critically involved in the definition of a component of a system suddenly leaves for another job, the rationale no longer is lost. Instead, the reasons why the component's requirements are stated as they are can be accessed within the IBIS framework. Rationale is no longer lost with employee turnover.

IBIS is a non-intrusive technique. A tool that changes the way people normally work may be viewed as intrusive or worse, and result in a decline in communication effectiveness [Yakemovic 90, p. 116]. Because IBIS is easy to understand and implement, it does not adversely affect the way meetings are carried out, and hence it supports the unobtrusive recording of important requirements information as it is communicated.

IBIS has a number of shortcomings as well, which are reported with welcome openness and honesty in [Yakemovic 90], [Conklin 88], and [Conklin 91]. A few of these problems are not specific to IBIS, but apply to all requirements elicitation documentation methods. For example,

it is noted that IBIS is vulnerable to "scheduling pressure", whereby all forms of documentation are abandoned in favor of getting code written [Yakemovic 90]. This can be stated against any form of requirements documentation.

IBIS does not support automated checking of attributes like consistency because there are not enough formalisms used in the method. The problem with extending IBIS to include more formalisms, though, is that more comprehensive techniques may not get used because of their increased complexity.

IBIS has no support for types outside of issues, positions, and arguments. Other types, such as goals and requirements, would be very useful for requirements elicitation, particularly if IBIS is used in conjunction with another methodology such as CORE. In addition, there is no support in IBIS for resolving issues and reaching consensus, i.e., for choosing among the various positions of an issue. All IBIS does is document the different positions. Some of these criticisms are addressed in [Ramesh 91], where the IBIS model is extended to provide primitives which do relate the argumentation process to the ramifications of the decisions made during that process. These extensions include decisions (which resolve issues), constraints which are generated by making decisions and which affect the design solution, and design objects, which when synthesized form the complete solution. Another solution would be to integrate IBIS with JAD. IBIS is well suited for documenting ideas, while JAD provides support for idea generation and group consensus.

IBIS does not have embedded support for the evolution of the issues-positions-arguments hierarchy. There is no easy way to identify which positions are old and perhaps superseded by newer positions, or which issues are visited the most frequently on traversals through the hierarchy. Since requirements are known to undergo iterative refinement, ideally a method for requirements elicitation would support this evolution.

Text-only tools for IBIS do not organize information well, particularly for larger projects. Tracking the issues-positions-arguments framework with text documents results in an unmanageable amount of data scattered across many files. When text-only tools were employed in this way, once the framework grew to a sufficient size users stopped updating the electronic versions of the files, because finding the relevant information became increasingly difficult. For this reason, the text-only tools were replaced by a graphically oriented hypertext tool called gIBIS. Such a tool had its own problems, listed below.

In order to be built with fewer resources, gIBIS was implemented on top of an existing relational database management system. As a result, though, gIBIS is a closed system. Nodes representing issues, positions, and arguments within IBIS cannot be linked adequately to material which is not a node. For example, a diagram of the evolving design cannot be linked to the issue from which that diagram was derived. Likewise, a prototype used to decide between two positions cannot be linked to those two positions. Such links would promote traceability.

The gIBIS hypertext tool forced ideas to be expressed in a fine-grained separated manner, which obscures the larger ideas being developed by the author [Conklin 88, p. 327]. Understanding the individual issues, positions, and arguments without the benefit of a larger context is difficult for the parties involved in requirements elicitation.

Some general hypertext research issues apply to gIBIS, i.e., hypertext support for IBIS, as well. For example, an open question is how a large body of information is best represented for ease of navigation. It is difficult for some users of gIBIS to traverse through a large number of nodes, and hence these users will not create frameworks that exceed the number of nodes with which they can comfortably examine.

A final disadvantage to capturing requirements rationale with IBIS is that it is difficult to organize the rationale and access a particular piece when needed [Yakemovic 90]. Therefore one of the key inhibitors to the use of IBIS is the need for better information organization, a point that was also brought up during the SEI requirements workshop [SEI 91].

A.2 Notes on the Use of Domain Models

Domain analysis is defined as the definition of features and capabilities common to systems in advance of software development, and is noted as providing a means of communication and a common understanding of the domain [Kang 90, p. 2]. As such domain analysis is of paramount importance in requirements elicitation, where those that understand the problem domain converse with those who need to understand it in order to capture the requirements effectively. The cited report on feature-oriented domain analysis (FODA) continues that the requirements analyst uses the products of domain analysis when implementing a new system [Kang 90, p. 5]. Therefore, FODA does have applicability to requirements elicitation, and will be overviewed in this section.

All requirements originate with the end-users and/or the buyers, i.e., decision-makers, of a system. These requirements are transcribed by a requirements analyst, and must be understandable and usable by the system designers. A useful requirements elicitation methodology addresses these communicative issues. In addition, a frame of reference is needed on which to base communication between the requirements analyst and the end-users and/or buyers. This is provided via domain knowledge captured from domain experts, existing systems, and literature concerning the domain. In order to be useful, such domain knowledge needs to be organized, and this organization shall be referred to as a domain model.

Domain information comes from textbooks, standards, existing applications, and domain experts [Kang 90, p. 26]. Depending on the domain analysis technique employed, more than one type of model could be generated from this information. For example, with FODA both an entity relationship model and features model are created. The entity relationship model is particularly useful for educating the requirements analyst about the domain, and the features model is particularly useful for communicating to the end user the issues and decisions involved in creating requirements for a system in this domain. Regardless of the number of component models

within the domain model, the domain model itself is likely to have issues which need to be resolved in order to specify an application within the domain. "Issues" are the descriptions of points within the domain model where alternative selections can be made. "Decisions" are descriptions of these alternative selections for issues, along with the rationale behind why this selection would be made over the others within this domain. The use of a domain model which identifies such issues and decisions for all alternative points would be extremely useful toward structuring requirements elicitation techniques in that domain.

Through the use of a domain model, the requirements analyst would interview end-users and buyers to both validate the domain model's appropriateness for this particular context, and to resolve the issues present within the model. This process is iterative in nature, i.e., through interviews with end-users it may become obvious that possible alternatives and corresponding issues and decisions exist at a point in the domain model where none had been previously noted. By returning back to the sources of domain knowledge, the analyst can verify the prescribed changes, and the domain model can be updated accordingly.

As the domain model evolves, so will its usefulness for gathering requirements for future applications within the domain. The economic justification for using a domain model during requirements elicitation are thus twofold. First, the application under development will be of a better quality and can be developed at a reduced cost because with a domain model, the issues and decisions that need to be made during the requirements phase will be clearly stated. Second, the reuse of prior domain analysis work will lead to reduced costs for future developments in the area.

The domain model serves three other main purposes besides structuring the communication between the end-users/buyers and requirements analyst. It defines the domain terminology for all parties to reduce communication ambiguity. It simplifies requirements conflict detection between different end-users/buyers by identifying issue points where such conflicts can arise. It also promotes requirements conflict resolution by providing the rationale behind the decisions which can be made at issue points.

The detection of conflicts is simplified by the identification of conflict-producing areas within the domain model, i.e., issues. The resolution of the conflicts is initiated by the analyst after retrieving all issues where two or more end-users or buyers gave inconsistent decisions. By using a domain model to structure the elicitation process, the requirements analyst will be kept at the domain, i.e., problem, level and not drop into the software design level.

There are a few disadvantages to the use of domain modeling techniques in requirements elicitation. The domain may not be mature enough to enable the development of a domain model sufficiently rich to improve the elicitation process. Even in mature domains, a domain model may not be readily available. In those cases, the development of a domain model may be viewed as too costly in terms of time or effort, especially if no further work in that domain is being planned. Finally, in cases where a domain model is available and is used as an aid to elicitation, the domain model potentially could be a limiting factor in the development of complete requirements. The system being elicited may only be an improper subset of the domain

covered by the domain model, with some of the system extending beyond the scope of the domain model. Users, developers, and the requirements analyst may be biased toward staying within the framework provided by the domain model, even if a certain need is better satisfied through an extension to the domain model.

The following paragraphs address how elicitation structured by a domain model exhibits the six traits of an ideal problem analysis technique outlined by Davis [Davis 90]:

1. **Facilitates communication.** By using a domain model, communication can be performed in a structured manner. The use of a domain terminology dictionary can clarify ambiguities. The use of issues and decisions can explain trade-offs to be made in specifying an application within a domain.

2. **Provides a means of defining the system boundary.** The domain model defines the boundary of a particular domain. For example, with FODA the context model is used for this purpose. If it can be verified that the system being elicited is a proper subset of the given domain, then the system boundary will be within this domain boundary. The boundary-setting procedures of the domain model can also be used to define the system boundary, which will likely be more restrictive.

3. **Provides a means of defining partitions, abstractions, and projections.** Different domain modeling techniques support one or more of these methods for structuring the domain. For example, the features model supports partitioning and abstraction. The domain model is the means by which the elicitation methodology supports these organizational approaches.

4. **Encourages analyst to think and document at the problem level, not the software level.** Again, the domain model should be of sufficiently broad scope so as to avoid software design issues, and the use of the domain model to structure elicitation will therefore keep the analyst at the problem level.

5. **Allows for opposing alternatives but alert the analyst to their presence.** The elicitation methodology could retain all the information gathered from different sources, including conflicts between two or more interviews. The use of "issues" marks the areas where conflicts may occur, easing the task of conflict identification.

6. **Makes it easy for the analyst to modify the knowledge structure.** As the problem is explored the original domain model used to structure this exploration may need to be modified in light of new information. The process of building the domain model and using that model to elicit requirements is iterative in nature. Therefore, the elicitation methodology realizes the need for modifying the knowledge structure, as applied to both the domain model and the preliminary requirements template. The intuitiveness of the domain model addresses ease of use. However, tool support is another important factor in determining how easy a technique is to use.

A.3 Notes on JAD

JAD is an acronym for Joint Application Design, a registered trademark of IBM Corporation [Wood 89]. JAD's main theme is to bring together representatives with management authority and accountability into a structured workshop to foster timely decision-making. JAD consists of five phases [Wood 89]:

1. project definition
2. research
3. preparation for the JAD session
4. the JAD session
5. the final document

Wood and Silver present a detailed description of these JAD phases in their book [Wood 89]. This book deals primarily with online transaction-based systems, especially those targeted for mainframes. There is an emphasis in the examples on data elements, screen flow, and screen design which does not apply equally to real-time systems, leading one to question whether this is a shortcoming of the book or whether JAD does not apply well to real-time systems.

The project definition and research phases of the JAD process deal with fact-finding and information gathering, two elicitation tasks mentioned earlier in the body of this report. Techniques which support these tasks, such as CORE, could likewise be used to improve the project definition and research done in preparation for a JAD session. The information gathered during the early phases of JAD is emphasized as what could work, not what "will be the way" of doing things [Wood 89, p. 101]. The JAD session is then used to validate the information gathered in the previous phases. The JAD process concentrates on this JAD session itself, and thus JAD contributes to requirements elicitation as primarily a means to validate information already gathered.

The make-up of the JAD team is crucial to the JAD session's success. The right people have to be involved, and the presence of a skilled facilitator can keep the session focused and can minimize unproductive emotional attacks and defenses. Having the right people involved allows decisions to be made quickly. The JAD session is also used to establish shared ownership in the final document (specification) among all the important stakeholders who should be present at the session. JAD provides "a concentrated workshop for making decisions with everyone present to make those decisions" [Wood 89, p. 147].

A JAD session can be costly, however, in terms of people and time. A JAD session may contain 18 to 25 people [Wood 89]. The JAD session is capable of producing agreement on "3 screens per half day session" [Wood 89, p. 111], which hints that a JAD session for a complex, real-time system may span many days.

Techniques and tools can improve the efficiency of the JAD session. For example, decision-making can be assisted through the use of structured techniques such as the Kepner-Tregoe weighted sums approach [Wood 89, p. 154]. Magnetics can be used for visualizing data elements, screen flow, and screen design. The use of prototyping for screen design is also mentioned by Wood and Silver as a communication mechanism, although the only prescriptive advice on using this prototyping technique is "don't spend too much effort here" since the design is only proposed, not finalized [Wood 89, p. 110].

While tools and techniques are useful in supporting JAD, Wood and Silver focused primarily on the JAD process, explicitly stating "first the methodology, then the tools" [Wood 89, p. 179]. The reader is left to speculate on how techniques and tools can further improve the JAD validation process. One example would be the treatment of issues raised during the JAD session. Using a technique such as IBIS would allow these issues to be documented with positions and arguments, fostering better consistency checking and providing a means for tracking rationale.

A.4 Notes on CORE

CORE, an acronym for COntrolled Requirements Expression, is a requirements analysis and specification method that was developed in 1979 and refined during the early 1980s [SDS 85]. It has not been used very much in practice in the U.S., despite being featured, or at least mentioned, in many research papers dealing with requirements elicitation, e.g., [Mullery 79], [Potts 91], [Kean 91]. Rome Air Development Center's (now Rome Laboratory's) work on the Requirements Engineering Environment (REE) includes the CORE method.

CORE can be considered as a paradigm in which the requirements specification is created by both the customer and the developer, not one or the other. It is based on the principle of first defining the problem to be analyzed, breaking it down into viewpoints to be considered, and gathering and documenting information about each viewpoint. This data is then further analyzed and structured, an enhanced graphical representation of the data for each viewpoint is created, and then the viewpoints are examined in combination rather than in isolation. The last phase of the method deals with constraints analysis, in which nonfunctional requirements such as cost and time windows are identified and the earlier phases are reviewed with respect to these nonfunctional requirements. For a complete description of the CORE method, consult the documentation from Systems Designers Scientific [SDS 85].

Some of the advantages of CORE are that it is a thoroughly documented method complete with a set of prescriptive guidelines on how to apply the method to a problem [SDS 85]. The method is a general purpose, flexible approach to requirements elicitation, allowing it to be applied to a broad class of problems (although, as will be discussed shortly, it does apply better to the earlier phases of development such as concept exploration rather than later phases such as detailed design).

CORE recognizes that requirements elicitation involves contributions from a number of different communities, a major point from the recent SEI requirements analysis workshop [SEI 91]. CORE defines the responsibilities of the members of these communities, e.g., Viewpoint Authorities, and structures the communication between these groups.

CORE has some known disadvantages. First, the role of the requirements analyst in CORE seems too passive. The assumptions are that the users will propose solutions and the analyst will make sense of them, and that the customer authority and not the analyst will resolve conflicting views of the proposed system. The analyst is not likely to be able to take such a passive role and end up with a coherent system requirements specification.

Second, the steps of the method are not very precise. Some concepts in CORE are not well defined, and the use of information gathered during certain phases of CORE is unclear.

Third, CORE is deficient in representing constraints, e.g., conditions under which an action is performed. It is also deficient in representing real-time requirements.

CORE does not provide any modeling primitives to support the expression of internal functions, e.g., with CORE you cannot decompose actions into subactions.

Finally, CORE does not provide the framework for storing the rationale behind the requirements with the requirements themselves. The rationale for any given viewpoint can be assumed to rest with the Viewpoint Authority, but as was pointed out in the SEI requirements workshop this individual or set of people is likely to change over time [SEI 91]. Therefore it is worthwhile to document the rationale for the requirements and to store such rationale with the requirements.

A.5 Notes on QFD

The term "quality" is defined in Japan Standard Z8101-1981 as "a system of means to economically produce goods or services which satisfy customer requirements " Quality Function Deployment (QFD) is "an overall concept that provides a means of translating customer requirements into the appropriate technical requirements for each stage of product development and production" [ASI 86, p. II-39]. The initial steps of QFD can be described as being "simply a system of identifying and prioritizing customer needs obtained from every available source" [ASI 86, p. VII-6]. QFD is a concept which applies well to requirements elicitation, especially to an elicitation model where the customer's voice is the driving force behind requirements creation. This section will first present a very brief overview of the QFD methodology, and then explore further the advantages and disadvantages of QFD, especially when applied to requirements elicitation. As QFD has been applied successfully to automobile manufacturing, and given that most readers are familiar with automobiles, examples of QFD applied to automobile manufacturing will be used to clarify points made in this section.

The traditional process model involved in automobile manufacturing was to start with the customer requirements, and through a product development process derive the product. QFD refines this process model into the following:

1. Start with the customer requirements.
2. Transform these requirements into design requirements.
3. Successively transform design requirements into part characteristics, then manufacturing operations, and finally production requirements.

Since the first few transformations are those which are most applicable to the requirements elicitation phase, the focus of these notes will be on the early planning phase the QFD methodology.

The planning phase of QFD can be decomposed into nine steps:

1. Identify the customers, the product or service under design, and the development time horizon.
2. Gather high-level customer requirements and refine each requirement into a set of actionable items. Requirements are expressions of what the system does which is perceptible and has value to customers. Sources of these requirements for automobile manufacturing include marketing research, dealer input, sales department wants, magazines, and surveys. Examples of customer requirements from the automobile domain include "easy to repair," "easy to service," "easy to close door," and "easy to clean."
3. Gather final product control characteristics that should be assured to meet customer requirements. These control characteristics are chosen by the designers to satisfy customer requirements. Examples from the automobile domain include wall thickness on doors, door lock location, door lock size, and door frame width. A correlation matrix can also be developed for these characteristics to show which characteristics are in conflict with one another.
4. Develop relationship matrix (strongly related, related, weakly related) between customer requirements and product control characteristics.
5. Add market evaluation (customer importance ratings and competitive evaluations).
6. Compute final product control characteristic competitive evaluation for each control characteristic. Then compare these control characteristic competitive evaluations to market competitive evaluations to determine any inconsistencies.
7. List the key selling points of the product.
8. Develop target values for each of the final product control characteristics based on the key selling points, customer importance ratings, and current product strengths and weaknesses.

9. Finalize selection of the product control characteristics to deploy through the remainder of the process (from planning through to production), dependent on the information used to develop target values as well as the feasibility of achieving the target values for those characteristics.

The QFD documents produced later in the process, including the Deployment Matrix, Process Plan and Control Chart, and Operating Instructions, will not be presented here since they apply less directly to requirements elicitation. These planning phase steps enumerated above, however, are analogous to many of the activities performed during requirements elicitation. Requirements elicitation can be presented as involving the following steps in order to clarify the potential relationship with the QFD method:

1. Identify the communities affected by the proposed system, e.g., customers, users, and developers. Also identify any initial constraints identified by the customer which affect requirements development.
2. Gather high-level requirements from the customer. Ideally this procedure is done in a structured manner, perhaps by first breaking the problem down into different viewpoints (via CORE method), and then gathering the requirements and decomposing them via the IBIS technique.
3. Derive and gather a number of potential system architecture features that need to exist in order to meet the customer requirements.
4. Develop the relationship matrix between the customer requirements and system architecture features.
5. Add the prioritization, i.e., customer importance ratings, of each of the customer requirements. Also, for commercial products, perform a competitive evaluation of systems meeting the customer's needs.
6. Compute final system architecture feature competitive evaluation for each architecture feature. Then compare these feature evaluations with the market competitive evaluations to determine any inconsistencies.
7. List the key customer requirements, i.e., the key "selling points."
8. Develop a target value for each of the final system architecture features.
9. Define a system architecture to use based on the key customer requirements, customer importance ratings, current systems' capabilities, and the feasibility of achieving the target values computed in the previous step.

A more careful examination of the relationship between QFD and requirements elicitation may reveal that QFD is ideal for defining the system architecture requirements, as hinted at here.

For more details on QFD, consult the various papers and presentations collected in [ASI 86]. Another reference on QFD is [Akao 90].

Some goals of QFD which are similar to the goals presented for requirements elicitation at the requirements engineering workshop sponsored by the SEI [SEI 91] are:

- educate management, especially middle management (with elicitation, management also needs to be educated on the importance of the elicitation phase and the benefits of promoting communication during this phase)
- capture the "voice of the customer," i.e., capture what the customer likes rather than the problem, allowing design decisions to be traced back to customer needs (with elicitation, the user community's views are crucial and should be captured, possibly through the use of mock-ups and prototypes, and the rationale behind these views should be saved)
- horizontal deployment, i.e., "group work" rather than "self-promoting" (communities involved in elicitation should work together to achieve the common goal of producing quality requirements and sharing ownership in these requirements)

QFD strengthens the current development process and achieves the desired output efficiently. It strengthens the development process by:

- defining clear targets early based on market and business demands;
- focusing simultaneously on product and process technologies;
- prioritizing the key issues for better resource allocation; and
- enhancing communication and teamwork.

It achieves the desired output efficiently by:

- meeting the customer's needs with the product; and
- providing a product which has a competitive edge.

Advantages of applying QFD to system requirements elicitation include:

- emphasizing designing for quality by focusing on the customer's needs
- promoting teambuilding
- improving cross-functional communication
- addressing high priority items early
- preserving knowledge in the QFD documents (promoting reuse)
- reducing cost through decreased start-up problems
- shortening product development time (in part by the virtual elimination of late engineering changes)
- enhancing design reliability
- increasing customer satisfaction

Disadvantages of applying QFD to system requirements elicitation include the following:

- More work must be done in the planning stages.
- It is more difficult to change direction once started, since a lot of work has been expended to follow a given direction and all interrelationships would have to be revisited and revised upon a change in direction.

- Many steps in the method outlined for QFD planning are applicable to the development of a competitive product, e.g., the use of market surveys and analysis of competitor's products. Some of these steps may not apply to government sole source contracts.
- The QFD method outlined in the different sections of [ASI 86] does not indicate the process by which the decomposed customer requirements and product control characteristics are derived. For example, it does not indicate how the design of an automobile is broken down into smaller parts, e.g., the door. Further, it does not specify how the particular product control characteristics such as "door lock size", and the particular customer requirements such as "easy to close door", are derived. Other methods (e.g., CORE) might be used in conjunction with QFD to describe how the decomposition of the problem space and architecture space should occur during requirements elicitation.
- The QFD method does not provide stopping conditions on the decomposition of customer requirements, i.e., the ideal granularity of customer requirements is not specified.

Despite these disadvantages, it appears upon a cursory reading of the QFD literature that there are many aspects of QFD which should be incorporated into a proposed methodology for requirements elicitation.

A.6 Notes on SSM

Both the definition of methodology and the philosophy behind soft systems methodology (SSM) are given as follows [Checkland 89b, p. 101]:

> *It is the essence of a* ***methodology****—as opposed to a method, or technique—that it offers a set of guidelines or principles which in any specific instance can be tailored both to the characteristics of the situation in which it is to be applied and to the people using the approach: users of SSM have to discover for themselves ways of using it which they personally find both comfortable and stimulating. ...Such is the variety of human problem situations that no would-be problem solving approach could be reduced to a standard formula and still manage to engage with the richness of particular situations. Flexibility in use is characteristic of competent applications of SSM, and the reader should not look for a handbook formula to be followed every time.*

SSM is applicable to "messy, changing, ill-defined problem situations" [Checkland 89a, p. 72]. While classic operations research starts with a carefully defined objective, in most managerial problems this question is part of the problem, i.e., "there are few human situations in which getting the logic right is enough to bring about action" [Checkland 89a, p. 77]. Thus SSM treats "what to do" as part of the problem. Requirements elicitation under SSM should focus not only on decision-making with respect to finding a solution to a problem, but also on problem formulation as well. It is a "soft" approach in that it takes into account multiple perspectives, including effectiveness (the right thing to do), efficacy (it works), and efficiency (minimum use of resources).

SSM is a learning, and not an optimizing, system. Learning has to be participative, i.e., SSM is not supposed to be the skill of some external expert [Checkland 89a, p. 98].

The seven phases of SSM are enumerated below. As pointed out earlier, though, SSM is not a static cookbook and these phases do not need to be progressed through linearly without iteration. The phases are as follows:

1. Enter the situation considered problematic. The problem is typically unstructured.
2. Express the problem situation. These first two steps together consist in "finding out."
3. Identify the essential objectives of the problem situation by formulating a root definition. This root definition contains or implies the six pieces of information represented by the mnemonic CATWOE: customers, actors, transformation process, Weltanshauung (world view or outlook), owner, and environmental constraints.
4. Create conceptual models of one or more systems to meet those objectives. These models take into account efficiency, efficacy, and effectiveness. Along with the root definition, these models form the systems thinking about the real world.
5. Compare the models to reality (expressed in the second steps). If inadequate, return to the root definition and revise.
6. Define possible changes which are both desirable and feasible.
7. Take action to improve the problem situation.

There are both advantages and disadvantages to SSM's general nature. Potential problems with the methodology can be argued away by claiming that SSM can be tailored to an organization's needs to better address any given problems. However, this general framework can be cited as a limitation because it may fail to offer enough prescriptive guidelines to get started with the methodology and use it during requirements elicitation.

SSM uses conceptual, idealized models to contrast desired states with actual states. This approach works fine for the development of a new or novel system. In other elicitation scenarios, however, such as one where a revision to an existing system is to be built, there is a significant cost associated with radically redesigning the actual, existing system. Rather than working from a desired state and attempting to attain it, a possibly more cost-effective and potentially more appealing approach would be to incorporate constraints from the existing system when defining the requirements. It is difficult, however, to identify when constraints should be kept from the existing system, versus when an evolution toward an ideal, desired state should be pursued regardless of the existing system. More information on SSM is provided in [Checkland 90].

A.7 Other Method-Specific Notes

This section contains disjoint notes concerning various requirements elicitation strategies. These strategies have not yet been detailed like IBIS in Appendix A.1 and CORE in Appendix A.4, nor have they been fully analyzed for the advantages and disadvantages they could offer if incorporated into a proposed elicitation methodology. This section is useful mainly for identifying references to other elicitation strategies.

Among the elicitation strategies found in the recent literature which are not the subject of a separate appendix section in this report are the following:

- Entity-Relationship-Attribute-Event (ERAE) [Dubois 88]
- User Skills Task Match (USTM), a user-centered approach to requirements expression [Macaulay 90]. USTM is structured into three stages of description (describing a product opportunity), analysis (identifying a high-value solution), and decision-making (delivering a business solution). It is a collection of techniques and methods designed for use by the key stakeholders in the development of initial requirements for "generic" systems. Generic systems are defined as those which are designed to satisfy the needs of many different customers or markets.
- Paisley [Zave 82], which views the requirements specification as an executable model of the proposed system interacting with its environment. Paisley models a system as a series of asynchronous interacting processes, but is criticized as being "a textual notation that is difficult to understand" [Deutsch 88, p. 41]. Paisley specifications are "operational" in that they are implementation-independent models of how to solve a problem, rather than just being statements of what properties a solution should have. More recent papers on Paisley such as [Zave 86] focus more on specification than with requirements elicitation and analysis, which were given mention in [Zave 82].
- Scenario Based Requirements Elicitation (SBRE) [Holbrook 90]. SBRE features the parallel development of requirements and a high level design, the use of scenarios to communicate the behavior of a design, and an evaluation function to assess the suitability of the design. SBRE also recognizes the importance of an issue base with which to maintain the issues that arise during the elicitation process, a point well addressed by IBIS and discussed in Appendix A.1. Scenarios are used to structure the early interaction between users and designers in order to quickly develop a set of initial requirements. The author suggests that because scenarios have low cost and limited expressiveness, they "seem most appropriate for communicating specific system features in situations of high uncertainty" [Holbrook 90, p. 97].
- Jackson System Development (JSD) [Cameron 86]. JSD involves specifications consisting mainly of a distributed network of sequential processes. It has been used primarily on data-processing applications. The major criticisms of JSD are that it is a very different "middle-out" approach, the design is performed in a very fragmented manner, and that it may not apply well to large systems since there is no hierarchical representation nor any overall view [Jackson 88]. JSD can be used to structure elicitation, but is often used at a later development stage [Cameron 86, p. 222]:

Many projects that have used JSD actually started slightly later in the life cycle, doing the first steps largely from existing documents rather than directly with the users.

- SCS (Structured Common Sense) [Finkelstein 86]. SCS is an formalized outgrowth of CORE and contains the following steps:
    a. Agent identification (based on CORE's viewpoint hierarchies)
    b. Physical data flow analysis (derived from structured systems analysis)
    c. Action tabulation and description (derived from CORE's tabular collection)
    d. ERA analysis
    e. Causal tabulation and special case analysis

  There were problems in trying to fully evolve SCS from existing methods [Finkelstein 86, p. 238]:

> *...no existing method appeared wholly suitable as a basis for producing a specification in the formal system. For example we adopt the philosophy of JSD, in which a system specification and design is build from a model of the envisaged system in its environment, but the process-oriented structure of the system and the procedural description of the processes is incompatible with our formal system. Similarly, the requirements elicitation heuristics of CORE and the notion of viewpoints and their actions fit very well into SCS and the formal system, although the subsequent dataflow representations of CORE do not.*

Bibliography

[ASI 86] American Supplier Institute. *Quality Function Deployment: A Collection of Presentations and QFD Case Studies.* American Supplier Institute, 1986.

[Congress 90] U.S. Congressional Subcommittee on Investigations and Oversight. *Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation.* Technical Report 4/90 Staff Study, U.S. 101st Congress, Washington, DC, April 1990.

[DoD 91] U.S. Department of Defense. *Software Technology Plan: Volume II Plan of Action* (Technical Report Draft 5, 8/15/91), U.S. Department of Defense, August 1991.

[IEEE 83] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology.* ANSI/IEEE Standard 729-1983, Institute of Electrical and Electronics Engineers, New York, 1983.

[IEEE 84] Institute of Electrical and Electronics Engineers. *IEEE Guide to Software Requirements Specifications.* ANSI/IEEE Standard 830-1984, Institute of Electrical and Electronics Engineers, New York, 1984.

[IEEE 90] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Standard 610.12-1990 (revision and redesignation of IEEE Std. 729-1983), Institute of Electrical and Electronics Engineers, New York, 1983.

[SDS 85] Systems Designers Scientific. *CORE: The Method.* Systems Designers Scientific, Camberley, England, 1985.

[SEI 91] Software Engineering Institute Requirements Engineering Project. *Requirements Engineering and Analysis Workshop Proceedings.* Technical Report CMU/SEI-91-TR-30 or ESD-TR-91-30, Software Engineering Institute, Pittsburgh, PA, December 1991.

[STEP 91] Software Test & Evaluation Panel (STEP), Requirements Definition Implementation Team. *Operational Requirements for Automated Capabilities*, Draft Pamphlet (Draft PAM), April 23, 1991.

[Akao 90] Akao, Yoji. *Quality Function Deployment: Integrating Customer Requirements into Product Design.* Productivity Press, Cambridge, MA, 1990.

[Andriole 90] Andriole, Stephen J. Command and Control Information Systems Engineering: Progress and Prospects. *Advances in Computers 31*:1-98, 1990.

[Ashworth 89] Ashworth, C. M. Using SSADM to Specify Requirements. In *IEE Colloquium on 'Requirements Capture and Specification for Critical Systems'* (Digest No. 138), 3/1-3/3. Institution of Electrical Engineers, November 1989.

[Bailin 89] Bailin, Sidney C. An Object-Oriented Requirements Specification Method. *Communications of the ACM 32*(5):608-623, May 1989.

[Begeman 88] Begeman, Michael L., and Conklin, Jeff. The Right Tool for the Job. *BYTE 13*(10):255-266, October 1988.

[Benyon 87] Benyon, D., and Skidmore, S. Towards a Tool Kit for the Systems Analyst. *The Computer Journal 30*(1):2-7, 1987.

[Berlin 89] Berlin, Lucy M. User-Centered Application Definition: A Methodology and Case Study. *Hewlett-Packard Journal 40*(5):90-97, October 1989.

[Brooks 87] Brooks, F.P. Jr. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 10-19, April 1987.

[Cameron 86] Cameron, John R. An Overview of JSD. *IEEE Transactions on Software Engineering SE-12*(2):222-240, February 1986.

[Cameron 89] Cameron, John R. Prototyping Core Functionality Using JSD. In *IEE Colloquium on 'Requirements Capture and Specification for Critical Systems'* (Digest No. 138), 2/1-2/2. Institution of Electrical Engineers, November 1989.

[Ceri 86] Ceri, Stefano. Requirements Collection and Analysis in Information System Design. In *Proceedings of the IFIP 10th World Computer Congress*, 205-214. IFIP World Computer Congress (10th: Dublin, Ireland), New York: North-Holland, September 1986.

[Checkland 89a] Checkland, Peter. Soft Systems Methodology. *Rational Analysis for a Problematic World.* New York: John Wiley & Sons, 71-100, Chapter 4, 1989.

[Checkland 89b] Checkland, Peter. An Application of Soft Systems Methodology. *Rational Analysis for a Problematic World.* New York: John Wiley & Sons, 101-119, Chapter 5, 1989.

[Checkland 90] Checkland, Peter & Scholes, Jim. *Soft Systems Methodology in Action.* New York: John Wiley & Sons, 1990.

[Colbert 89] Colbert, Edward. The Object-Oriented Software Development Method: A Practical Approach to Object-Oriented Development. In *TRI-Ada '89 Proceedings*, 400-415. ACM/SIGAda, October 1989.

[Conklin 88] Conklin, Jeff, and Begeman, Michael L. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems 6*(4):303-331, October 1988.

[Conklin 91] Conklin, Jeff, and Yakemovic, K.C. Burgess. A Process-Oriented Approach to Design Rationale. *Human-Computer Interaction6(3/4)*:357-391, 1991.

[Cordes 89] Cordes, D.W., and Carver, D.L. Evaluation Method for User Requirements Documents. *Information and Software Technology 31*(4):181-188, May 1989.

[Davis 90] Davis, Alan M. *Software Requirements: Analysis and Specification.* Prentice Hall: Englewood Cliffs, NJ, 1990.

[Delugach 91] Delugach, Harry S. *A Multiple Viewed Approach to Software Requirements.* Ph.D. thesis, University of Virginia, 1991.

[Deutsch 88] Deutsch, Michael S. Focusing Real-Time Systems Analysis on User Operations. *IEEE Software 5*(5):39-50, September 1988.

[Dobson 91] Dobson, J.E., Martin, M.J., Olphert, C.W., and Powrie, S.E. Determining Requirements for CSCW: the ORDIT Approach. *IFIP Conference on Collaborative Work, Social Communications and Information Systems (COSCIS91)*: 333-35. Elsevier Science Publishers B.V. (North-Holland), 1991.

[Dobson 92] Dobson, J.E., Blyth, A.J.C., Chudge, J., and Strens, M.R. The ORDIT Approach to Requirements Identification. *Proceedings of the Sixteenth Annual International Computer Software & Applications Conference.* IEEE Computer Society, September 1992.

[Dubois 88] Dubois, E., Hagelstein, J., and Rifaut, A. Formal Requirements Engineering with ERAE. *Philips Journal of Research 43*(3/4): 393-414, 1988.

[Fickas 88] Fickas, Stephen, and Nagarajan, P. Critiquing Software Specifications. *IEEE Software 5*:37-47, November 1988.

[Finkelstein 86] Finkelstein, Anthony, and Potts, Colin. Structured Common Sense: The Elicitation and Formalization of System Requirements. *Software Engineering 86.* London: Peter Peregrinus, Chapter 16, 1986.

[Finkelstein 88] Finkelstein, A. Re-Use of Formatted Requirements Specifications. *Software Engineering Journal 3*(5):186-197, September 1988.

[Fowler 90] Fowler, C.J.H., Kirby, M.A.R., and Macaulay, L.A. Historical Analysis: A Method for Evaluating Requirements Capture Methodologies. *Interacting with Computers 2*(2):190-204, August 1990.

[Goodrich 90] Goodrich, Victoria, and Olfman, Lorne. An Experimental Evaluation of Task and Methodology Variables for Requirements Definition Phase Success. In Bruce D. Shriver (editor), *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, 201-209. IEEE Computer Society, January 1990.

[Hackathorn 88] Hackathorn, R.D., and Karimi, J. A Framework for Comparing Information Engineering Methods. *Management Information Systems Quarterly 12*(2):203-220, June 1988.

[Holbrook 90] Holbrook, Capt. Hilliard III. A Scenario-Based Methodology for Conducting Requirements Elicitation. *ACM SIGSOFT Software Engineering Notes 15*(1):95-104, January 1990.

[Jackson 88] Jackson, Ken. Providing for the Missing Steps. *UNIX Review 6*(11):55-63, November 1988.

[Jokela 90] Jokela, Timo, and Lindberg, Kai. Statecharts Based Requirements Analysis: Deriving User Oriented Models. *Microprocessing and Microprogramming 30*(1-5):289-296, August 1990.

[Jordan 89] Jordan, Pamela W., Keller, Karl S., Tucker, Richard W., and Vogel, David. Software Storming: Combining Rapid Prototyping and Knowledge Engineering. *IEEE Computer*, 39-48, May 1989.

[Kang 90] Kang, Kyo C., Cohen, Sholom G., Hess, James A., Novak, William E., and Peterson, A. Spencer. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21, ADA235785, Software Engineering Institute, Pittsburgh, PA, November 1990.

[Kean 91] Kean, Elizabeth S. An Informal Approach to Developing an Environment for Requirements Capture and Refinement. In *Proceedings of the Requirements Engineering and Analysis Workshop Held 3/91*

(CMU/SEI-91-TR-30). Software Engineering Institute, December 1991.

[Kramer 87] Kramer, Jeff, Ng, Keng, Potts, Colin, and Whitehead, Ken. *Tool Support for Requirements Analysis.* Research Report DoC 87/3, Imperial College of Science and Technology, London, March 1987.

[Kramer 88] Kramer, J., Ng, K., Potts, C., and Whitehead, K. Tool Support for Requirements Analysis. *Software Engineering Journal 3*(3):86-96, May 1988.

[Lavi 88] Lavi, J.Z., and Winokur, M. Embedded Computer Systems: Requirements Analysis and Specification—An Industrial Course. In Ford, G.A. (editor), *Proceedings of the 1988 SEI Software Engineering Education Conference*, 81-105. Software Engineering Institute, Springer-Verlag, April 1988.

[Leite 87] Leite, Julio Cesar S P. *A Survey on Requirements Analysis.* Advanced Software Engineering Project Technical Report RTP-071, University of California at Irvine, Department of Information and Computer Science, June 1987.

[Loucopoulos 89] Loucopoulos, P., and Champion, R.E.M. Knowledge-Based Support for Requirements Engineering. *Information and Software Technology 31*(3):123-135, April 1989.

[Macaulay 90] Macaulay, Linda, Fowler, Chris, Kirby, Mark, and Hutt, Andrew. USTM: A New Approach to Requirements Specification. *Interacting with Computers 2*(1):92-118, April 1990.

[Mays 85] Mays, R.G., Orzech, L.S., Ciarfella, W.A., & Phillips, R.W. PDM: A requirements Methodology for Software System Enhancements. *IBM Systems Journal 24*(2):134-149, 1985.

[McDermid 89] McDermid, J. A. Requirements Analysis: Problems and the STARTS Approach. In *IEE Colloquium on 'Requirements Capture and Specification for Critical Systems'* (Digest No. 138), 4/1-4/4. Institution of Electrical Engineers, November 1989.

[Milsom 89] Milsom, F.D. Using CORE with SDL to Specify Requirements. In *Proceedings of the Seventh International Conference on Software Engineering for Telecommunication Switching Systems: SETSS 89* (Conf. Publ. No. 306), 137-141. Institution of Electrical Engineers (IEE), London, 1989.

[Mittermeir 90] Mittermeir, Roland T., Roussopoulos, Nicholas, Yeh, Raymond T., and Ng, Peter A. An Integrated Approach to Requirements Analysis. *Modern Software Engineering: Foundations and Current Perspectives.* New York: Van Nostrand Reinhold, 119-164, Chapter 5, 1990.

[Mullery 79] Mullery, G.P. CORE: A Method for Controlled Requirements Specification. In *Proceedings of the 4th International Conference on Software Engineering,* 126-135. IEEE Computer Society Press, 1979.

[Mullery 89] Mullery, G.P. Method Engineering: Methods via Methodology. In *IEE Colloquium on 'Requirements Capture and Specification for Critical Systems'* (Digest No. 138), 1/1-1/3. Institution of Electrical Engineers, November 1989.

[Mumford 81] Mumford, E. Participative Systems Design: Structure and Method. *Systems, Objectives, Selections 1*(1): 5-19, 1981.

[Nosek 88] Nosek, John T., and Schwartz, Ruth B. User Validation of Information System Requirements: Some Empirical Results. *IEEE Transactions on Software Engineering 14*(9):1372-1375, September 1988.

[Potts 91] Potts, Colin. Seven (Plus or Minus Two) Challenges for Requirements Analysis Research. In *Proceedings of the Requirements Engineering and Analysis Workshop Held 3/91* (CMU/SEI-91-TR-30). Software Engineering Institute, December 1991.

[Ramesh 91] Ramesh, B. & Dhar, V. Representation and Maintenance of Process Knowledge for Large Scale Systems Development. In *Proceedings of the Sixth Annual Knowledge-Based Software Engineering Conference,* 288-299. Rome Laboratory, Griffiss AFB, NY, September 1991.

[Rzepka 89] Rzepka, William E. A Requirements Engineering Testbed: Concept, Status, and First Results. In Bruce D. Shriver (editor), *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences,* 339-347. IEEE Computer Society, 1989.

[Sage 90] Sage, Andrew P., and Palmer, James D. *Software Systems Engineering.* New York: John Wiley & Sons, 1990.

[Schouwen 90] van Schouwen, A. John. *The A-7 Requirements Model: Re-Examination for Real-Time Systems and an Application to Monitoring Systems.* Technical Report 90-276, Queen's University, May 1990.

[Schubert 89] Schubert, M.A. Quality Function Deployment: 'A Comprehensive Tool for Planning and Development.' In *Proceedings of the IEEE 1989 National Aerospace and Electronics Conference NAECON 1989* (Cat. No. 89CH2759-9), 1498-1503. IEEE Dayton Section Aerospace and Electronic Systems Society, IEEE, New York, May 1989.

[Southwell 87] Southwell, K., James, K., Clarke, B.A., Andrews, B., Ashworth, C., Norris, M., and Patel, V. (Requirements Specification authoring team). Requirements Definition and Design. *The STARTS Guide, Second Edition*, Volume I. National Computing Centre, 177-313, Chapter 5, 1987.

[Stair 91] Stair, Ralph M. Jr., and LaMothe, Richard S. The Use of Systems Planning Methodologies. *Journal of Computer Information Systems 31*(2):34-37, Winter, 1990-1991.

[Stephens 85] Stephens, M., and Whitehead, K. The Analyst—A Workstation for Analysis and Design. In *Proceedings of the Eighth IEEE International Conference on Software Engineering*. IEEE Computer Society Press, 1985.

[Weinberg 88] Weinberg, Gerald M. *Rethinking Systems Analysis and Design.* New York: Dorset House Publishing, 1988.

[Winokur 90] Winokur, M., Lavi, J.Z., Lavi, I., and Oz, R. Requirements Analysis and Specification of Embedded Systems Using ESCAM—A Case Study. In *Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering* (Cat. No. 90CH2867-0), 80-89. IEEE Computer Society Press, May 1990.

[Wood 89] Wood, Jane & Silver, Denise. *Joint Application Design: How to Design Quality Systems in 40% Less Time.* New York: Wiley, 1989.

[Yakemovic 90] Yakemovic, K.C. Burgess, and Conklin, E. Jeffrey. Report on a Development Project Use of an Issue-Based Information System. In *CSCW '90 Proceedings.* ACM, October 1990.

[Zahniser 90] Zahniser, Richard A. How to Speed Development with Group Sessions. *IEEE Software*, 109-110, May 1990.

[Zave 82] Zave, P. An Operational Approach to Requirements Specification for Embedded Systems. *IEEE Transactions on Software Engineering SE-8*(3):250-269, May 1982.

[Zave 86] Zave, P., and Schell, W. Salient Features of an Executable Specification Language and Its Environment. *IEEE Transactions on Software Engineering SE-12*(2):312-325, February 1986.

[Zeroual 89] Zeroual, K. An Approach for Automating the Specification-Acquisition Process. In *Proceedings of the Second International Workshop on Software Engineering and Its Applications*, 349-355. Toulouse '89, EC2, Nanterre, France, 1989.

[Zucconi 89] Zucconi, Lin. Techniques and Experiences Capturing Requirements for Several Real-Time Applications. *ACM SIGSOFT Software Engineering Notes 14*(6):51-55, October 1989.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-92-TR-12	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESC-TR-92-012
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (city, state, and zip code) ESC/AVS Hanscom Air Force Base, MA 01731
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office 8b. OFFICE SYMBOL (if applicable) ESC/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO: 63756E; PROJECT NO.: N/A; TASK NO: N/A; WORK UNIT NO.: N/A
11. TITLE (include security classification) Issues in Requirements Elicitation	
12. PERSONAL AUTHOR(S) Michael G. Christel, Kyo C. Kang	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM TO	14. DATE OF REPORT (year, month, day) September 1992 15. PAGE COUNT 78
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	requirements engineering, system development, requirements elicitation, elicitation, specification, validation

19. ABSTRACT (continue on reverse if necessary and identify by block number)

There are many problems associated with requirements engineering, including problems in defining the system scope, problems in fostering understanding among the different communities affected by the development of a given system, and problems in dealing with the volatile nature of requirements. These problems may lead to poor requirements and the cancellation of system development, or else the development of a system that is later judged unsatisfactory or unacceptable, has high maintenance costs, or undergoes frequent changes. By improving requirements elicitation, the requirements engineering process can be improved, resulting in enhanced system requirements and

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution
22a. NAME OF RESPONSIBLE INDIVIDUAL Tom Miller, Lt Col, USAF	22b. TELEPHONE NUMBER (include area code) (412) 268-7631 22c. OFFICE SYMBOL ESC/AVS (SEI)

ABSTRACT —continued from page one, block 19

potentially a much better system. Requirements engineering can be decomposed into the activities of requirements elicitation, specification, and validation. Most of the requirements techniques and tools today focus on specification, i.e., the representation of the requirements. This report concentrates instead on elicitation concerns, those problems with requirements engineering that are not adequately addressed by specification techniques. An elicitation methodology is proposed to handle these concerns. This new elicitation methodology strives to incorporate the advantages of existing elicitation techniques while comprehensively addressing the activities performed during requirements elicitation. These activities include fact-finding, requirements gathering, evaluation and rationalization, prioritization, and integration. Taken by themselves, existing elicitation techniques are lacking in one or more of these areas.